LOCALIZACIÓN ON-LINE DE PLAZAS DE APARCAMIENTO EN UN ENTORNO URBANO

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC)

Autor: David Rojo González

Director: Prof. Ulises Cortés

Titulación: Enginyeria Informàtica Superior

Departament de Llenguatges i Sistemes Informàtics

ÍNDICE

1. INTRODUCCIÓN

La búsqueda de plazas de aparcamiento públicas supone un problema en muchas ciudades. Éstas han tenido un gran crecimiento y siguen conservando vías de circulación originales y muchas veces muy antiguas que no se han podido adaptar al aumento de usuarios que poseen vehículos privados. Debido a esto, en algunas zonas concretas de una ciudad, encontrar aparcamiento se convierte en una tarea que puede durar mucho tiempo, con el consiguiente gasto innecesario de combustible y exceso de contaminación.

Además, las plazas públicas de aparcamiento disponen de diversas restricciones de uso, como puedan ser vados horarios, zonas verdes y azules, carga y descarga o acceso para minusválidos que reducen las plazas disponibles para cada tipo de vehículo.

Una parte del tráfico que discurre por las ciudades son vehículos buscando plazas de aparcamiento. Éstos vehículos circulan a menor velocidad circulando continuamente por una misma zona entorpeciendo el tráfico.

La detección de plazas libres de aparcamiento está ya implementada en párquines de centros comerciales y aeropuertos. Esto se debe, a que al ser plazas privadas perfectamente delimitadas, se pueden controlar de forma eficiente. Al disponer de cientos de plazas en un espacio reducido es primordial evitar que los vehículos entorpezcan el tráfico hasta encontrar una plaza libre.

Esta detección de plazas libres en párquines privados reduce significativamente los vehículos que están circulando en la zona ya que cada conductor sabe de antemano si en un pasillo existen plazas o no, con lo que se dirige directamente a la zona de plazas libres indicadas en los paneles informativos.

Esta misma tecnología se puede aplicar a la vía pública obteniendo los mismos resultados y ventajas multiplicando su efecto reductor en el consumo de tiempo y combustible.

1. **Reducción del tráfico.** Debido a que los vehículos se dirigen directamente a una plaza libre, estos están menos tiempo circulando por las calles.

2. **Reducción de pequeños accidentes.** Los conductores que están buscando plazas de aparcamiento, están más pendientes de encontrar una plaza libre que del resto de elementos del tráfico, con lo que pueden provocar pequeños accidentes que a su vez entorpecen más el tráfico.

3. **Reducción de emisiones contaminantes.** Cuanto más tiempo esté un vehículo en circulación buscando una plaza de aparcamiento, mayor será el consumo de combustible y su contaminación.

4. **Ahorro económico.** Una reducción de tiempo en la búsqueda de aparcamiento supone además del ahorro de combustible un ahorro de tiempo para los trabajadores que estén desarrollando una actividad económica, con ello podrán dedicar más tiempo a su actividad y aumentará su productividad.

1.1. ANÁLISIS DE IMPACTO ECONÓMICO Y AMBIENTAL

En este apartado se va a poner en perspectiva y con números el impacto del ahorro de tiempo en la búsqueda de aparcamiento tanto a nivel económico como ambiental para una ciudad como Barcelona.

Tiempo medio para encontrar aparcamiento	20min[1]
Número de plazas de aparcamiento sin regulación	160.000[2]
Número de vehículos que buscan aparcamiento en plazas sin regulación al día	**320.000** *2 vehículos por plaza al día*
Número de plazas de aparcamiento en zona azul	9.000[3]

[1] Tiempo medio de búsqueda de aparcamiento en el mundo según un estudio realizado por IBM en 2011: http://www-03.ibm.com/press/es/es/pressrelease/35510.wss

[2] http://revista.consumer.es/web/es/20080201/actualidad/tema_de_portada/72320_5.php#barcelona

[3] Barclona de Serveis Municipals S.A, Inform d'auditoria independent, Comptes anuals al 31 de desembre de 2015 i Informe de gestió de l'exercici 2015 pàgina 69, https://www.bsmsa.cat/fileadmin/user_upload/transparencia/informacio_economica/comptes_anuals/Comptes_anuals_2015_signades.pdf

Número de vehículos que buscan aparcamiento en zona azul al día	**45.000** *Rotación de 5 vehículos al día*
Número de plazas de aparcamiento en zona verde	39.500
Número de vehículos que buscan aparcamiento en zona verde al día	**79.000** *2 vehículos por plaza al día*
Número total de vehículos buscando aparcamiento al día	**444.000** vehículos/día
Tiempo total buscando aparcamiento	**~148.000** horas
Kilómetros totales recorridos en búsqueda de aparcamiento	**1.480.000** Km *Velocidad media de 10km/h*
CO2 ~160gr/Km	**237 toneladas/día**
CO ~0.5gr/Km	**0,74 toneladas/día**
Combustible total consumido	**88.800 litros/día** *Consumo medio de 6l/100km*
Ahorro económico en combustible	Gasolina: 61.000€ *50%4 de vehículos a 1,37€/l5* Gasoil: 52.000 € *50% de vehículos a 1,13€/l*
Ahorro económico en salarios	Sueldo: 925.000 € *Renta media de 12.000€* *Media de 8h por jornada*
Ahorro económico total	**1.038.000 €/día**

[4] Según DGT parque automovilístico de turismos en Barcelona en 2014 tabla 4.1: http://www.dgt.es/es/seguridad-vial/estadisticas-e-indicadores/parque-vehiculos/tablas-estadisticas/2014/

[5] Precio medio de la gasolina y el diésel en Barcelona en el año 2015 http://www.dieselogasolina.com/Estadisticas/Historico

Con estos datos tenemos una ahorro económico anual de casi de **250 millones de euros**, un **0,39% del PIB de Barcelona**[6].

En el ámbito medio-ambiental, el ahorro sería de casi **57.000 toneladas de monóxido de carbono** y **177 toneladas de dióxido de carbono** al año, lo que supone una reducción de un 1,58% de emisiones[7].

Hay que aclarar que el ahorro económico derivado de los sueldos, no sería un ahorro directo, ya que los trabajadores no dejarían de percibir la parte proporcional de su sueldo por el ahorro de tiempo, pero podrán dedicar ese tiempo a realizar su tarea profesional aumentando su productividad. También hay que tener en cuenta que no todo el mundo que busca aparcamiento está realizando una actividad profesional. Por lo tanto, este cálculo es una puesta en valor del ahorro de tiempo de los ciudadanos.

[6] Para un PIB de 64.241,5 millones de euros de 2015 de Barcelona: http://www.idescat.cat/emex/?id=080193&lang=es#h2000000

[7] Para 3,6 millones de toneladas emitidas por Barcelona en 2012: http://www.lavanguardia.com/local/barcelona/20151123/30348981646/barcelona-reduir-co2-contaminacion-compromiso-clima.html

2. OBJETIVOS

El objetivo final del proyecto es desarrollar un prototipo de sistema de búsqueda de plazas de aparcamiento realista para una ciudad como Barcelona, que incluya todo el apartado de gestión de las zonas a monitorizar y la búsqueda activa de las mismas por parte de un cliente desde un dispositivo móvil, así como una simulación de tráfico para poder ejecutar pruebas sobre la plataforma desarrollada.

Se utilizarán componentes *Software Libre* para el desarrollo de todos los componentes de la plataforma.

3. MOTIVACIÓN PERSONAL

Como usuario habitual de vehículo privado en una ciudad como Barcelona, he vivido en varias ocasiones el problema de búsqueda de aparcamiento. En mi caso perdía 3 horas al día en transporte público para ir al trabajo, luego ir a comer y luego ir a la universidad y volver a casa. El uso del vehículo privado me permitió ahorrar 1 hora y media al día pudiéndola dedicarla a tareas más productivas.

Aunque realmente ahorraba tiempo, me daba cuento que podría ahorrar mucho más tiempo si fuese más fácil encontrar aparcamiento allá donde fuera. En muchas ocasiones perdía más tiempo en encontrar una plaza libre que en el desplazamiento.

De ésta experiencia, he descubierto que la mejor manera es parar el coche en una zona segura y esperar a que alguien libere una plaza, pero esta operación no se puede efectuar en todos los sitios, por lo que la mayoría de ocasiones debes seguir circulando y tomar decisiones sobre qué dirección tomar sin ningún tipo de información. Si al tener que tomar una dirección, se supiera de antemano si hay una plaza libre o por el contrario, si no hay ninguna plaza libre, se ahorraría mucho tiempo.

Saber dónde se encuentra una plaza libre te permite ir directamente a ella. Por el contrario, saber que no hay ninguna plaza libre evita maniobras innecesarias, pudiendo parar el vehículo en algún sitio que no perjudique la circulación a la espera de una plaza libre, o directamente aparcar en un parking privado y así sacar el vehículo de la circulación.

Considero que esta herramienta puede ayudar mucho a una ciudad con de tráfico y de aparcamiento, mejorando la calidad de vida de sus ciudadanos.

Este proyecto me permitirá aplicar los conocimientos aprendidos durante la carrera de Ingeniería en Informática y la práctica personal. Por un lado me permitirá aplicar los conocimientos adquiridos en Ingeniería del Software para especificar la aplicación, por otro lado, los conocimientos adquiridos en sistemas y redes me servirán en la definición de comunicaciones entre componentes, definición de arquitectura e implementación del servidor. Asimismo, los conocimientos de programación, algoritmia y estructura de datos me permitirán abordar el desarrollo e implementación de todos los componentes del proyecto.

Aunque tengo experiencia personal y profesional en el desarrollo de aplicaciones en entorno web y sobre todo en el lenguaje *PHP*[8], para este proyecto quiero emplear nuevas tecnologías que al empezar el proyecto desconozco.

El uso del framework *Symfony*[9] es nuevo para mí, aunque lo he tratado por encima en alguna ocasión no he desarrollado nada serio con él, por lo que este proyecto me permitirá profundizar en su uso y su ecosistema. Asimismo, desconozco el ecosistema de desarrollo de aplicaciones en *Android*, con lo que aunque conozco bastante bien el lenguaje *Java*, la programación en *Android* es nueva para mí.

También desconozco el uso de bases de datos *NoSQL*, y en concreto la base de datos *MongoDB*[10] con la que se trabajará en el proyecto.

En resumen, la mayor parte de las herramientas para desarrollar el proyecto son desconocidas para mí, y el proyecto me obligará a estudiarlas y dominarlas para poder realizarlo.

[8] http://php.net/
[9] http://symfony.com/
[10] https://www.mongodb.com/

4. ESTADO DEL ARTE

Hoy en día, la implantación de *Smartphones* sustituyendo los teléfonos móviles es imparable. Estos dispositivos en su mayoría permiten el acceso a Internet y la detección de la ubicación del usuario mediante GPS en tiempo real.

Por este motivo, el uso de GPS en el vehículo ha sido prácticamente sustituido por el *Smartphone*, permitiendo al usuario acceder a la misma información y en tiempo real sin un desembolso extra. Además le permite obtener mayor información que con un GPS clásico ya que estos dispositivos permiten fusionar información proveniente de distintas fuentes de una manera eficiente.

Los usuarios están acostumbrados a utilizar el *Smartphone* para ir a un lugar concreto de la ciudad, pero a la hora de aparcar éste deja de ser de utilidad y el usuario debe buscar su plaza de aparcamiento sin ningún tipo de ayuda circulando reiteradamente por la misma zona o aparcar en un parking privado. Nuestra idea es construir una app que amplíe las capacidades de un *Smartphone* para ayudar a encontrar una plaza de libre.

4.1.SOLUCIONES ADOPTADAS

Existen diversas formas de crear un mapa de plazas libres y ocupadas de aparcamiento para una región:

4.1.1. Aplicaciones colaborativas

Las aplicaciones colaborativas se basan en que unos usuarios avisan a otros usuarios del estado de las plazas de aparcamiento. Esta comunicación puede ser manual o automática, siendo el usuario el que de forma activa indique que ha dejado libre una plaza o la propia aplicación mediante *bluetooth* y *gps* detecte que el vehículo se ha movido o ha aparcado e indicarlo automáticamente.

- **Ventajas:** Permite la implementación del sistema sin ningún tipo de inversión en infraestructura, con lo que se puede aplicar a cualquier zona.
- **Desventajas:** Requiere que la persona que aparque o deje una plaza libre disponga de la aplicación, en caso contrario esa plaza no es monitorizada. Además de no

detectar cualquier obstrucción en la plaza de aparcamiento temporal (sacos de runas, obras, etc.)

4.1.2. Sensores individuales

En esta categoría entrarían los sensores que pueden detectar si un vehículo está estacionado en una plaza concreta. Estos sensores pueden estar ubicados sobre la plaza o en el suelo y se suelen encontrar en párquines privados con detección de plazas.

- **Ventajas:** Se puede calcular el número exacto de plazas disponibles.
- **Desventajas:** Supone un alto gasto en infraestructura para la implantación de los sensores. El mantenimiento de los sensores es complicado debido a su ubicación.

4.1.3. Sensores generales.

En esta categoría entrarían los sensores que pueden detectar varios vehículos en una zona concreta. Éstos, pueden ser cámaras de video junto con procesamiento de imágenes o radares.

- **Ventajas:** Son fáciles de instalar y de bajo coste debido a que se adaptan a cualquier tipo y forma de vía y un único sensor puede gestionar una calle completa.
- **Desventajas:** Son altamente complejos.

4.2. APLICACIONES EN EL MERCADO

Hoy en el mercado se encuentran varias aplicaciones para la búsqueda de aparcamiento, pero no hay ninguna que realmente solucione el problema perfectamente o esté lo suficientemente implantada para ofrecer a los usuarios una herramienta fiable.

4.2.1. Streetline

http://www.streetline.com/ California

- Aplicación colaborativa y procesamiento de imágenes de cámaras de vigilancia.
- Permite el pago de uso de plazas de aparcamiento mediante le móvil.
- Detección de matrículas.
- Analíticas de uso de las plazas de aparcamiento.

- Filtrado de plazas públicas/privadas y minusválidos/eléctrico/estudiantes.
- Implantado en: San Carlos (California), San Mateo (California), Universidad de Oregón.

4.2.2. Parkhelp

http://www.parkhelp.com/ Barcelona

- Aplicación mediante sensores unitarios.
- Publicación de plazas libres mediante paneles públicos.
- Implantado mayoritariamente en párquines privados.

4.2.3. FastPrk

http://www.fastprk.com/ Barcelona

- Aplicación mediante sensores individuales en el suelo
- Aplicación móvil que muestra disponibilidades por zonas pero no plazas concretas.
- Implantado en pequeñas zonas en Italia, Polonia, República Checa y Suiza

4.2.4. Wazypark

http://www.wazypark.com/ Madrid

- Aplicación colaborativa para Smartphones.
- Implantado en España

5. ANÁLISIS DE REQUISITOS

5.1. REQUISITOS FUNCIONALES

Los requisitos funcionales son aquellos requisitos que ha de cumplir el conjunto de aplicaciones a desarrollar para cumplir los objetivos.

- **Creación de plazas de aparcamiento**: Se necesita un sistema que permita gestionar toda la información de plazas disponibles de aparcamiento en una zona.
- **Creación de restricciones a las plazas de aparcamiento**: Se necesita un sistema que permita añadir restricciones a las plazas de aparcamiento, como pueden ser vados, zonas azules y verdes[11], etc.
- **Actualización de plazas libres**: Se necesita un sistema que detecte y actualice el estado de las plazas libres definidas automáticamente.
- **Consulta de plazas de aparcamiento**: Se necesita un sistema que permita consultar el estado de las plazas libres de una zona concreta.

5.2. REQUISITOS NO FUNCIONALES

Los requisitos no funcionales son aquellos requisitos que no son funcionalidades principales de la aplicación pero que deben cumplirse para su correcto funcionamiento.

- **Actualizaciones en tiempo real**: La actualización en la consulta de plazas libres debe ser en tiempo real. En cuanto una plaza queda libre u ocupada, el cliente que está visualizando dicha plaza debe ser notificado del cambio al momento.
- **Facilidad de gestión**: El sistema debe facilitar al máximo la consulta, introducción y actualización de datos. Al trabajar con posiciones geográficas, se utilizaran números muy parecidos constantemente, con lo que el sistema debe facilitar la visualización de la información para evitar errores.
- **Seguridad**: Se debe proteger el acceso a los diferentes servicios que el sistema exponga, controlando en todo momento que el acceso está garantizado.
- **Escalabilidad**: Se debe poder escalar el sistema para aumentar el número de peticiones y consultas.

[11] Las zonas verdes y azules se refieren a plazas de parking con distintos requisitos de permanencias y precio.

6. DISEÑO

Para cumplir los objetivos y requisitos de la aplicación a desarrollar se necesitarán los siguientes elementos:

- Un **sistema de gestión** que permita configurar y definir las diferentes plazas de aparcamiento disponibles en diversas zonas geográficas, permitiendo añadir restricciones a dichas plazas.

- Un **sistema de detección** de plazas de aparcamiento que informe del estado de cada plaza de aparcamiento definida.

  Para este proyecto, no se desarrollará ningún sistema de detección de plazas físicas, y se sustituirá por un **simulador de tráfico** que genere una simulación en una zona de vehículos que liberen y ocupen plazas de aparcamiento.

- Un **sistema de consulta** en formato de aplicación móvil que permita a un usuario consultar esta información y que determine las plazas disponibles a partir de las restricciones que defina el usuario, como tamaño del vehículo y zonas restringidas donde puede o no puede aparcar.

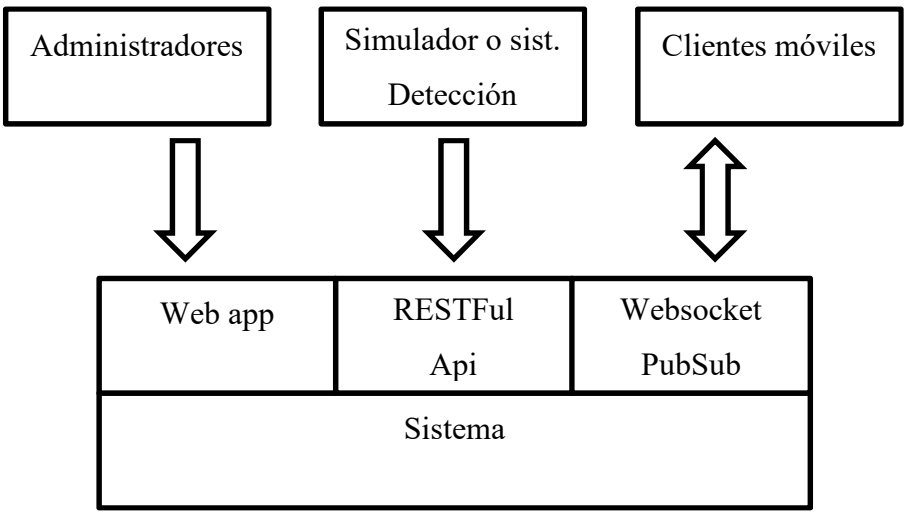
6.1. ARQUITECTURA

Los tres puntos a desarrollar son sistemas independientes que se deben comunicar entre ellos.

Para el sistema de gestión se empleará una **aplicación web**.

Para el **simulador de tráfico** se desarrollará como un script ejecutable que se comunicará con el sistema mediante el uso de una **API Restful**.

Para el sistema de consulta se empleará una **aplicación en Android** que se comunicará con el sistema mediante un **servidor PubSub** utilizando el estándar WAMP (Web Application Messaging Protocol) permitiendo así la comunicación en tiempo real entre el servidor y los clientes.



Se utilizará *MongoDB* como base de datos para almacenar toda la información referente a la búsqueda de plazas de aparcamiento. La motivación de utilizar este sistema es debido a:

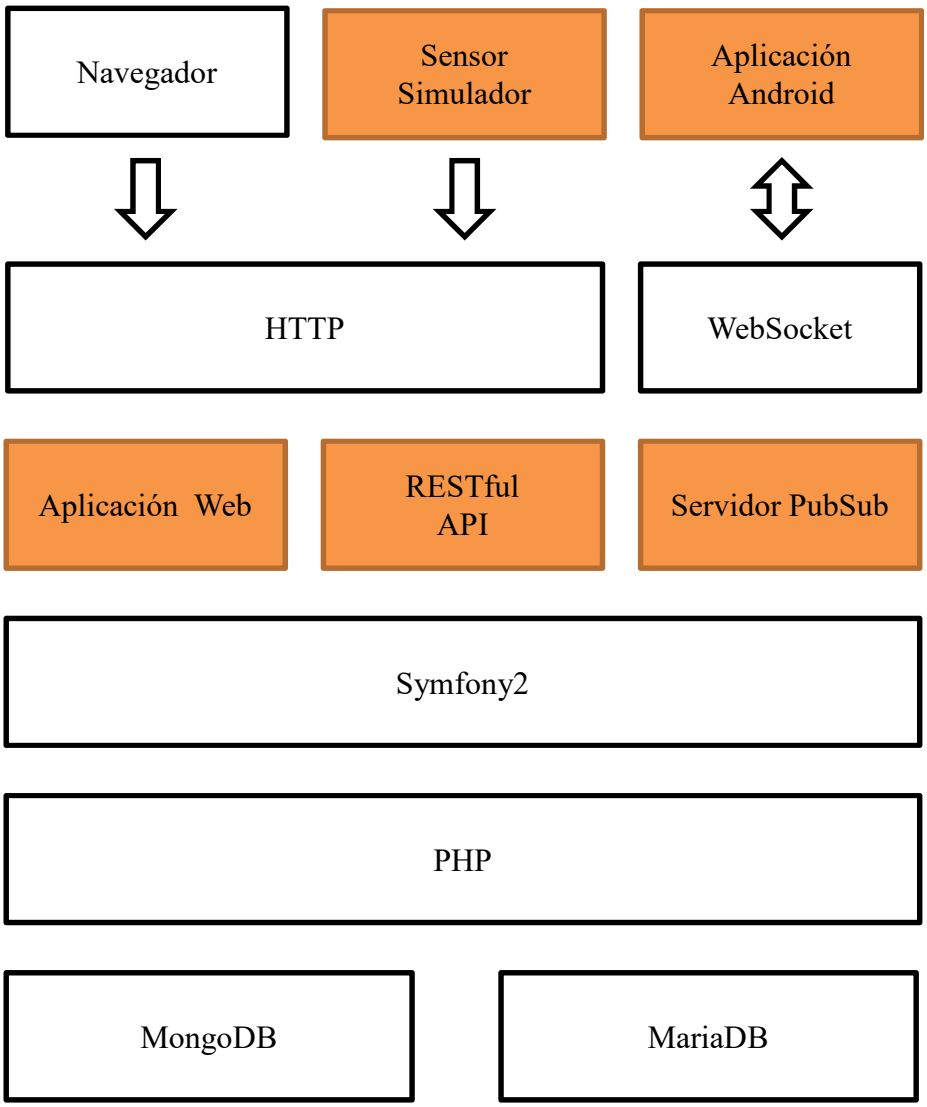
- Es una base de datos muy rápida.
- No es necesario el uso de consultas complejas.
- El sistema de documentos embebidos de MongoDB se adapta perfectamente a la base de datos a desarrollar.

La aplicación web, la API y el servidor PubSub se desarrollaran mediante *PHP* y el framework *Symfony2*, que también permitirá el desarrollo del simulador de tráfico.

La aplicación móvil se desarrollará mediante el lenguaje *Java* para *Android*.

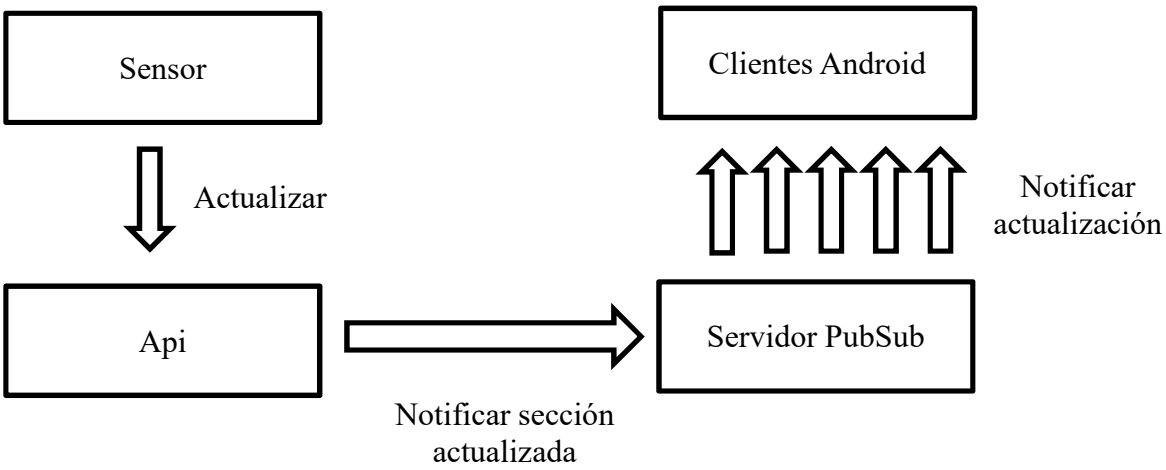
Componente	**Lenguaje**	**Framework**	**Otros**
Aplicación web	PHP	Symfony2	HTML, Javascript
API RESTful	PHP	Symfony2	
Servidor PubSub	PHP	Symfony2	
Simulador	PHP	Symfony2	
Cliente móvil	Java		Android SDK

Resumen general de la arquitectura. Los componentes de color naranja son los componentes desarrollados en este proyecto:



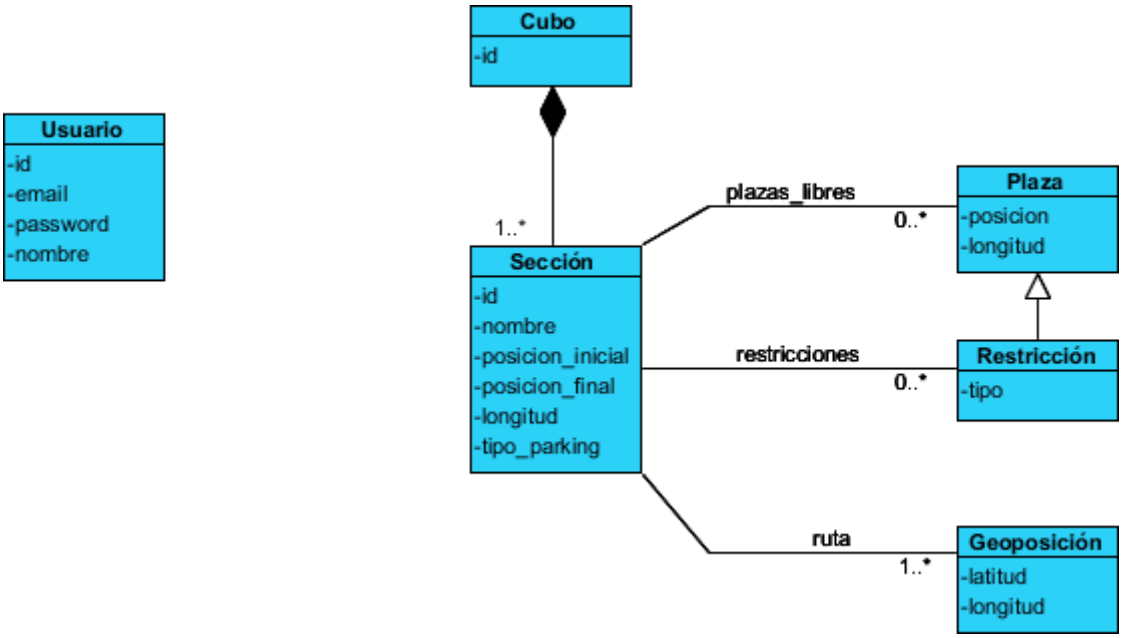
La comunicación entre los sensores y la aplicación Android se realizará mediante *Websockets*. La *API* recibirá las actualizaciones de los sensores y notificará al servidor *PubSub* de las secciones modificadas. El servidor al recibir la notificación consultará el estado de la sección actualizada y la comunicará a los clientes Android conectados.

Los clientes *Android* se subscribirán a regiones geográficas, recibiendo así todas las actualizaciones de dichas regiones. Cuando el servidor *PubSub* reciba una notificación de sección actualizada, deberá detectar a que región pertenece la sección y notificar a todos los clientes suscritos a dicha región los cambios en la sección.


Sensor
Actualizar
Api
Notificar sección actualizada
Servidor PubSub
Clientes Android
Notificar actualización

7. ESPECIFICACIÓN

7.1. MODELO CONCEPTUAL DE DATOS



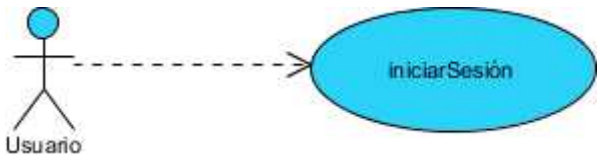
7.2. ACTORES

Se definen cinco tipos de actores en el sistema a desarrollar. Cada actor está asociado a cada uno de los componentes a desarrollar. Algunos actores pueden interactuar con diversos componentes.

7.2.1. Aplicación Web

7.2.1.1. Usuario

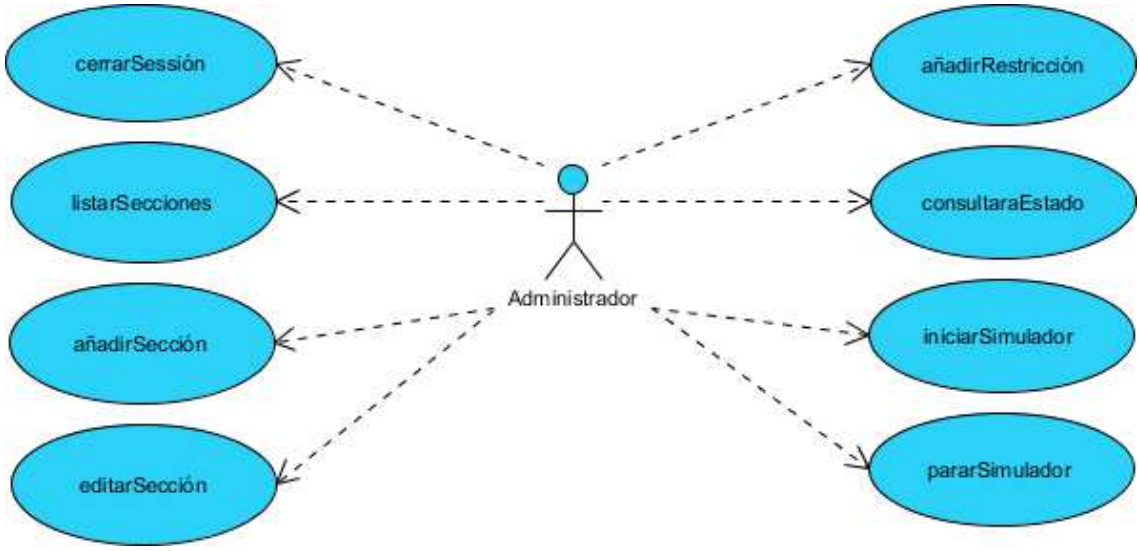
Un usuario es cualquier usuario que accede a la página web de administración. Los casos de uso de un Usuario son exclusivos de la aplicación web



La forma de interactuar del usuario con el sistema es mediante un navegador web y el protocolo HTTP.

7.2.1.2. Administrador

Un administrador es un usuario que se ha identificado correctamente en el sistema de administración. Los casos de uso de un Administrador son exclusivos de la aplicación web.

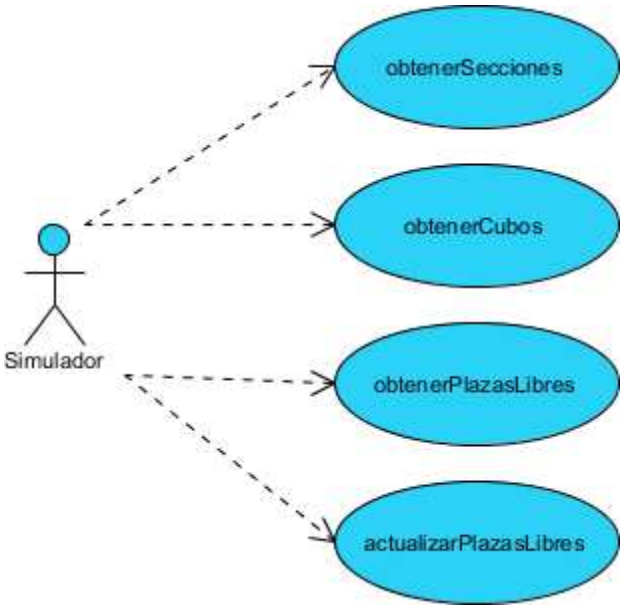


La forma de interactuar del administrador con el sistema es mediante un navegador web y el protocolo HTTP.

7.2.2. Simulador

7.2.2.1. Simulador

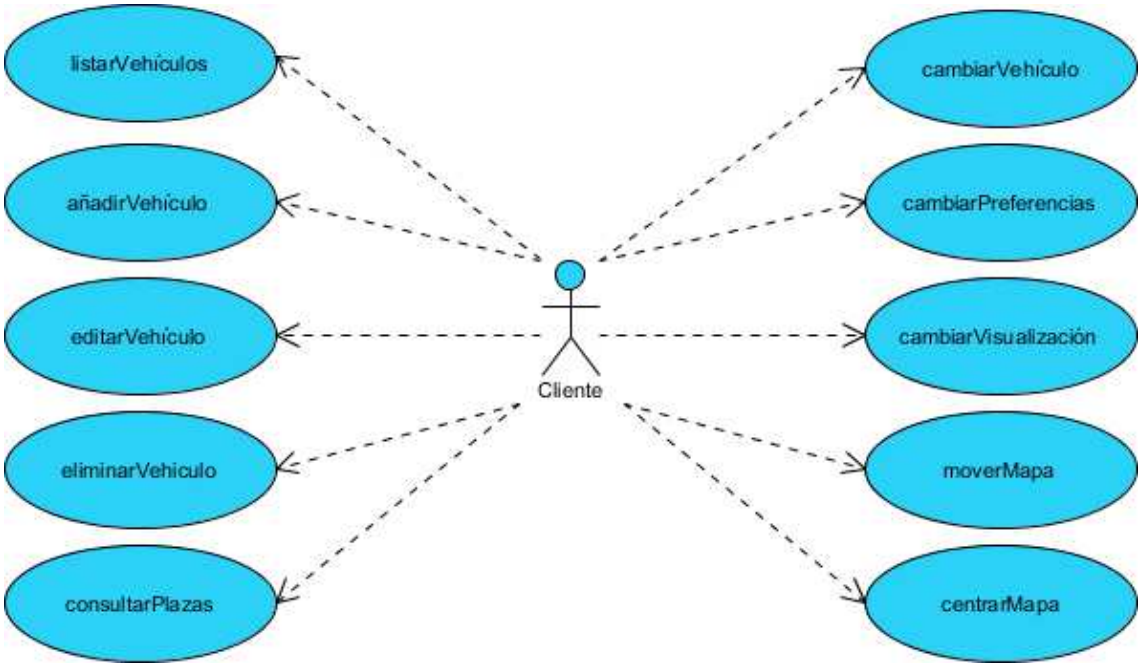
El simulador es el encargado de simular las operaciones que realizaría el sensor, por lo tanto tiene los mismos acceso que un sensor añadiendo algunos privilegios para poder realizar su labor. Los casos de uso de un simulador/sensor son exclusivos de la *API*.



7.2.3. Aplicación Móvil

7.2.3.1. Cliente

Un cliente es cualquier usuario que utilice la aplicación móvil para buscar aparcamiento. Los caso de uso de un cliente para la aplicación móvil.

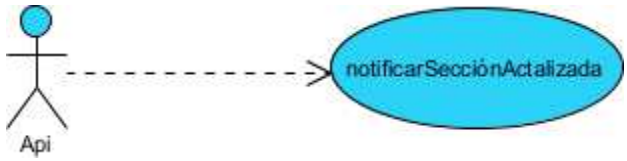


La forma de interactuar del cliente en con el sistema es mediante una aplicación en un dispositivo móvil.

7.2.4. Servidor PubSub

7.2.4.1. Api

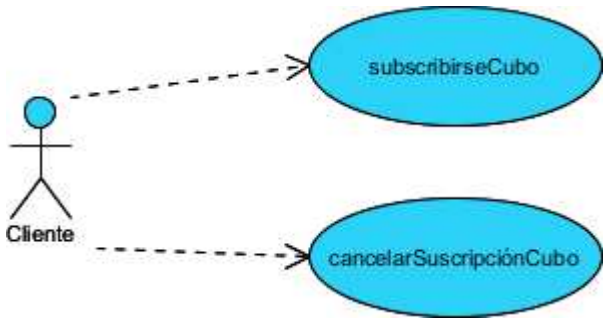
La Api puede interactuar con el servidor *PubSub* para notificarle cambios en el estado del sistema.



La forma de interactuar de la *API* con el servidor *PubSub* es mediante notificaciones *Push.*

7.2.4.2. Cliente

Un cliente puede utilizar el servidor *PubSub* para suscribirse o borrarse a actualizaciones de plazas libres.



La forma de interactuar del cliente con el servidor *PubSub* es mediante *WebSockets*.

7.3. CASOS DE USO

7.3.1. Aplicación web

7.3.1.1. inicarSesión

Nombre	iniciarSesión
Descripción	Inicia la sesión del usuario

Actores	Usuario
Precondiciones	
Curso típico de eventos	El usuario introduce su email y contraseñas y hace clic en el botón para iniciar sesión. El sistema comprueba que el email existe y la contraseña introducida coincide. El sistema concede da acceso al usuario.
Postcondiciones	El usuario pasa a ser un Administrador

7.3.1.2. cerrarSesión

Nombre	cerrarSesión
Descripción	Inicia la sesión del usuario
Actores	Administrador
Precondiciones	
Curso típico de eventos	El administrador hace clic en cerrar sesión. El sistema elimina su sesión
Postcondiciones	El usuario pasa a ser un Usuario.

7.3.1.3. listarSecciones

Nombre	listarSecciones
Descripción	Muestra las secciones existentes en el sistema.
Actores	Administrador
Precondiciones	
Curso típico de eventos	El usuario accede al listado de secciones. El sistema muestra un mapa centrado en la región activa. El sistema muestra las secciones de la región activa en un

	listado y en el mapa.
Postcondiciones	

7.3.1.4. añadirSección

Nombre	añadirSección
Descripción	Añade una nueva sección al sistema
Actores	Administrador
Precondiciones	
Curso típico de eventos	El usuario hace clic en añadir sección. El sistema muestra un formulario con los datos requeridos y un mapa para dibujar la sección. El usuario introduce los datos requeridos y dibuja la sección en el mapa. El usuario hace clic en guardar sección. El sistema añade la nueva sección. El sistema lleva al usuario al caso de uso de editarSección de la nueva sección añadida
Postcondiciones	Se crea una nueva sección en el sistema

7.3.1.5. editarSección

Nombre	editarSección
Descripción	Edita los datos de una sección
Actores	Administrador
Precondiciones	La sección existe
Curso típico de eventos	El usuario hace clic en editar una sección. El sistema muestra el formulario con los datos requeridos y un mapa con la sección dibujada.

	El usuario edita los datos o mueve la sección en el mapa. El usuario hace clic en guardar cambios. El sistema guarda la sección con los nuevos datos introducidos. El sistema vuelve a mostrar el caso de uso de editarSección de la sección editada.
Postcondiciones	Se actualiza la sección actual

7.3.1.6. añadirRestrición

Nombre	Añadir restricción
Descripción	Añade una restricción a una sección
Actores	Administrador
Precondiciones	La sección existe
Curso típico de eventos	El usuario hace clic en el botón de añadir restricción. El sistema muestra el formulario con los datos requeridos y un mapa con la vista previa de la restricción. El usuario introduce los datos de la restricción. El sistema actualiza la vista previa a medida que se introducen los datos. El usuario hace clic en guardar cambios. El sistema añade la restricción a la sección con los datos introducidos. El sistema vuelve a mostrar el caso de uso de editarSección de la sección previa.
Postcondiciones	Se añade la restricción a la sección.

7.3.1.7. consultarEstado

Nombre	consultarEstado
Descripción	Consulta el estado de las plazas libres.

Actores	Administrador
Precondiciones	
Curso típico de eventos	El usuario hace clic en consultar estado. El sistema muestra un mapa centrado en la ubicación actual con todas las secciones disponibles en rojo y con las plazas libres en verde. El sistema actualiza el estado del mapa en tiempo real tal como se vayan liberando y actualizando plazas.
Postcondiciones	

7.3.1.8. iniciarSimulador

Nombre	iniciarSimulador
Descripción	Inicia el simulador de tráfico
Actores	Administrador
Precondiciones	El simulador no está en ejecución.
Curso típico de eventos	El usuario hace clic en el botón de iniciar simulador. El sistema inicia el proceso de simulación de tráfico. El sistema muestra que el simulador está en ejecución.
Postcondiciones	El simulador está en ejecución.

7.3.1.9. pararSimulador

Nombre	pararSimulador
Descripción	Finaliza el simulador de tráfico
Actores	Administrador
Precondiciones	El simulador está en ejecución.
Curso típico de	El usuario hace clic en el botón de parar simulador.

eventos	El sistema finaliza el proceso de simulación de tráfico. El sistema muestra que el simulador no está en ejecución.
Postcondiciones	El simulador no está en ejecución.

7.3.2. API

7.3.2.1. obtenerSecciones

Nombre	obtenerSecciones
Descripción	Obtiene el listado de secciones dentro del rectángulo definido por dos puntos geográficos.
Actores	Administrador, Simulador
Precondiciones	
Curso típico de eventos	El usuario proporciona dos puntos geográficos. El sistema retorna una lista de secciones dentro del rectángulo definido por los dos puntos geográficos.
Postcondiciones	

7.3.2.2. obtenerCubos

Nombre	obtenerCubos
Descripción	Obtiene el listado de cubos dentro del rectángulo definido por dos puntos geográficos y sus adyacentes.
Actores	Administrador, Simulador
Precondiciones	
Curso típico de eventos	El usuario proporciona dos puntos geográficos. El sistema retorna una lista de cubos dentro del rectángulo definido por los dos puntos geográficos y sus cubos adyacentes.

Postcondiciones	

7.3.2.3. obtenerPlazasLibres

Nombre	obtenerPlazasLibres
Descripción	Obtiene el listado de plazas libres de una sección.
Actores	Simulador
Precondiciones	
Curso típico de eventos	El usuario proporciona la sección. El sistema retorna la lista de plazas libres de la sección.
Postcondiciones	

7.3.2.4. actualizarPlazasLibres

Nombre	obtenerPlazasLibres
Descripción	Actualiza el listado de plazas libres de una sección.
Actores	Simulador
Precondiciones	
Curso típico de eventos	El usuario proporciona la sección y un listado de plazas libres. El sistema actualiza las plazas libres de la sección. El sistema retorna la lista actualizada de plazas libres de la sección.
Postcondiciones	

7.3.3. Servidor PubSub

7.3.3.1. notificarSecciónActualizada

Nombre	notificarSecciónActualizada

Descripción	Notifica a los clientes que ha habido un cambio en una sección
Actores	Sistema
Precondiciones	La sección existe
Curso típico de eventos	El sistema notifica que una sección se ha actualizado. El servidor PubSub localiza el cubo al que pertenece la sección. El servidor PubSub obtiene los datos actualizados de la sección. El servidor PubSub notifica a todos los suscritos del cubo los datos actualizados de la sección.
Postcondiciones	Los clientes suscritos reciben la sección actualizada.

7.3.3.2. suscribirseCubo

Nombre	suscribirseCubo
Descripción	Añade al cliente a la lista de suscripciones de un cubo
Actores	Cliente
Precondiciones	
Curso típico de eventos	El cliente solicita la suscripción a un cubo. El servidor obtiene el cubo solicitado. Si el cubo existe, añade al cliente a la lista de suscripciones del cubo.
Postcondiciones	El cliente pasa a ser un suscriptor del cubo.

7.3.3.3. cancelarSuscripciónCubo

Nombre	cancelarSuscripciónCubo
Descripción	Elimina al cliente a la lista de suscriptores de un cubo

Actores	Cliente
Precondiciones	El cliente está suscrito al cubo solicitado.
Curso típico de eventos	El cliente solicita la cancelación de su suscripción a un cubo. El servidor elimina al cliente de la lista de suscriptores del cubo.
Postcondiciones	El cliente es eliminado de la lista de suscriptores del cubo.

7.3.4. Aplicación móvil

7.3.4.1. listarVehículos

Nombre	listarVehículos
Descripción	Lista los vehículos introducidos
Actores	Cliente
Precondiciones	
Curso típico de eventos	El usuario accede a la sección de listado de vehículos. El sistema muestra el listado de vehículos creados por el usuario.
Postcondiciones	

7.3.4.2. añadirVehículo

Nombre	añadirVehículo
Descripción	Lista los vehículos introducidos
Actores	Cliente
Precondiciones	
Curso típico de eventos	El usuario hace clic en el botón de añadir vehículo. El sistema muestra un formulario con los datos requeridos. El usuario introduce los datos del vehículo.

	El usuario hace clic en el botón de guardar cambios. El sistema guarda los datos del vehículos. El sistema muestra el caso de uso listarVehiculos.
Postcondiciones	Se añade un nuevo vehículo.

7.3.4.3. editarVehículo

Nombre	editarVehículo
Descripción	Actualiza los datos de un vehículo
Actores	Cliente
Precondiciones	El vehículo existe.
Curso típico de eventos	El usuario hace clic en el botón de editar vehículo. El sistema muestra un formulario con los datos requeridos con la información del vehículo a editar. El usuario edita los datos del vehículo. El usuario hace clic en el botón de guardar cambios. El sistema actualiza los datos del vehículos. El sistema muestra el caso de uso listarVehiculos.
Postcondiciones	Se actualizan los datos del vehículo.

7.3.4.4. eliminarVehículo

Nombre	eliminarVehículo
Descripción	Elimina un vehículo existente.
Actores	Cliente
Precondiciones	El vehículo existe.
Curso típico de eventos	El usuario hace clic en el botón de eliminar vehículo. El sistema solicita confirmación del usuario. El usuario hace clic en el botón de confirmar.

	El sistema elimina el vehículo. El sistema muestra el caso de uso listarVehiculos
Postcondiciones	Se elimina el vehículo.

7.3.4.5. consultarPlazas

Nombre	consultarPlazas
Descripción	Consulta las plazas libres de la región
Actores	Cliente
Precondiciones	El cliente tiene al menos un vehículo.
Curso típico de eventos	El usuario hace clic en el botón de consultar plazas. El sistema muestra un mapa centrado en la posición actual del usuario. El sistema mueve y orienta el mapa según el usuario se mueva. El sistema selecciona el último vehículo utilizado. El sistema muestra en el mapa las plazas libres para el vehículo seleccionado según la visualización por defecto. El sistema actualiza el mapa según se actualicen las plazas libres.
Postcondiciones	

7.3.4.6. cambiarVehículo

Nombre	cambiarVehículo
Descripción	Cambia el vehículo para el que se quiere buscar una plaza.
Actores	Cliente
Precondiciones	El cliente tiene al menos un vehículo.
Curso típico de	El usuario hace clic en el botón de cambiar vehículo.

eventos	El sistema muestra un listado de los vehículos existentes del usuario. El usuario selecciona un nuevo vehículo. El sistema actualiza las plazas libres para el nuevo vehículo.
Postcondiciones	El sistema cambia el vehículo predefinido del usuario.

7.3.4.7. cambiarPreferencias

Nombre	cambiarPreferencias
Descripción	Cambia las preferencias del usuario para buscar plaza.
Actores	Cliente
Precondiciones	
Curso típico de eventos	El usuario hace clic en el botón de cambiar preferencias. El sistema muestra un formulario con las preferencias de búsqueda de aparcamiento El usuario introduce sus nuevas preferencias. El sistema actualiza las plazas libres para las nuevas preferencias.
Postcondiciones	

7.3.4.8. cambiarVisualización

Nombre	cambiarVisualización
Descripción	Cambia la visualización para mostrar las plazas libres.
Actores	Cliente
Precondiciones	
Curso típico de eventos	El usuario hace clic en el botón de cambiar visualización. El sistema cambia la forma en que muestra las plazas libres (líneas o puntos).

Postcondiciones	Se actualiza la visualización por defecto.

7.3.4.9. moverMapa

Nombre	moverMapa
Descripción	Desplaza, inclina o se amplía el área de visualización de plazas libres.
Actores	Cliente
Precondiciones	
Curso típico de eventos	El usuario desplaza, inclina o amplia el área de visualización de plazas libres. El sistema actualiza el área de visualización y muestra el botón de centrar mapa. El sistema deja de actualizar la posición del mapa a medida que el usuario se mueve.
Postcondiciones	Se deja de actualizar el mapa cuando el usuario se mueve y se muestra el botón de centrar mapa.

7.3.4.10. centrarMapa

Nombre	moverMapa
Descripción	Desplaza, inclina o se amplía el área de visualización de plazas libres.
Actores	Cliente
Precondiciones	El usuario ha movido el mapa.
Curso típico de eventos	El usuario hace clic en el botón de centrar mapa. Se actualiza el mapa a la posición y dirección del usuario con el zoom por defecto. Se oculta el botón de centrar mapa.

	Se actualiza el mapa automáticamente según se mueve el usuario.
Postcondiciones	Se actualiza el mapa según se mueve el usuario y desaparece el botón de centrar mapa.

7.4. DIAGRAMAS DE SECUENCIA

A continuación se muestran algunos de los diagramas de secuencia más destacados. Se han seleccionado los diagramas de secuencia que aporten información acerca de la interacción entre los componentes desarrollados obviando aquellos que son simples operaciones básicas de creación, edición o listados de información.

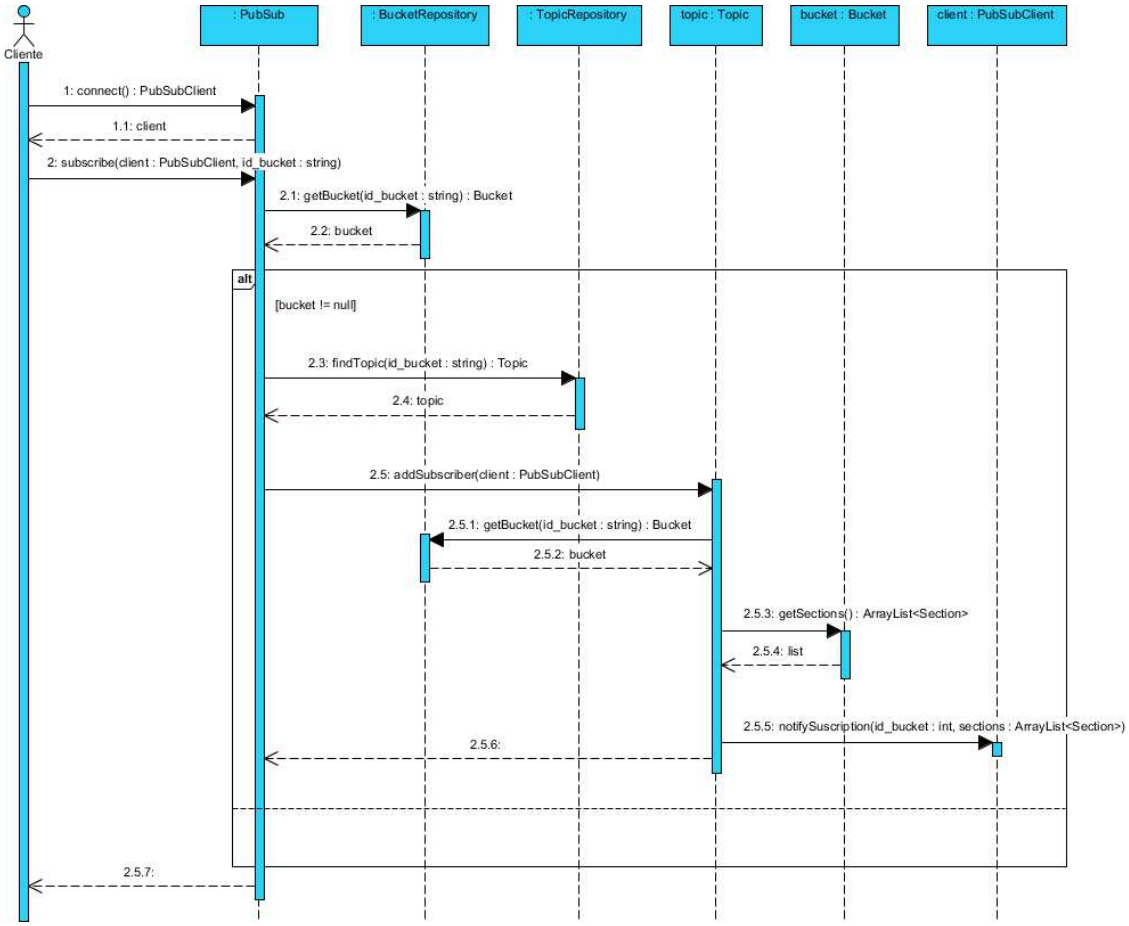
Para el desarrollo de las aplicaciones se utilizará el modelo MVC[12].

7.4.1. Aplicación web

7.4.1.1. consultarEstado

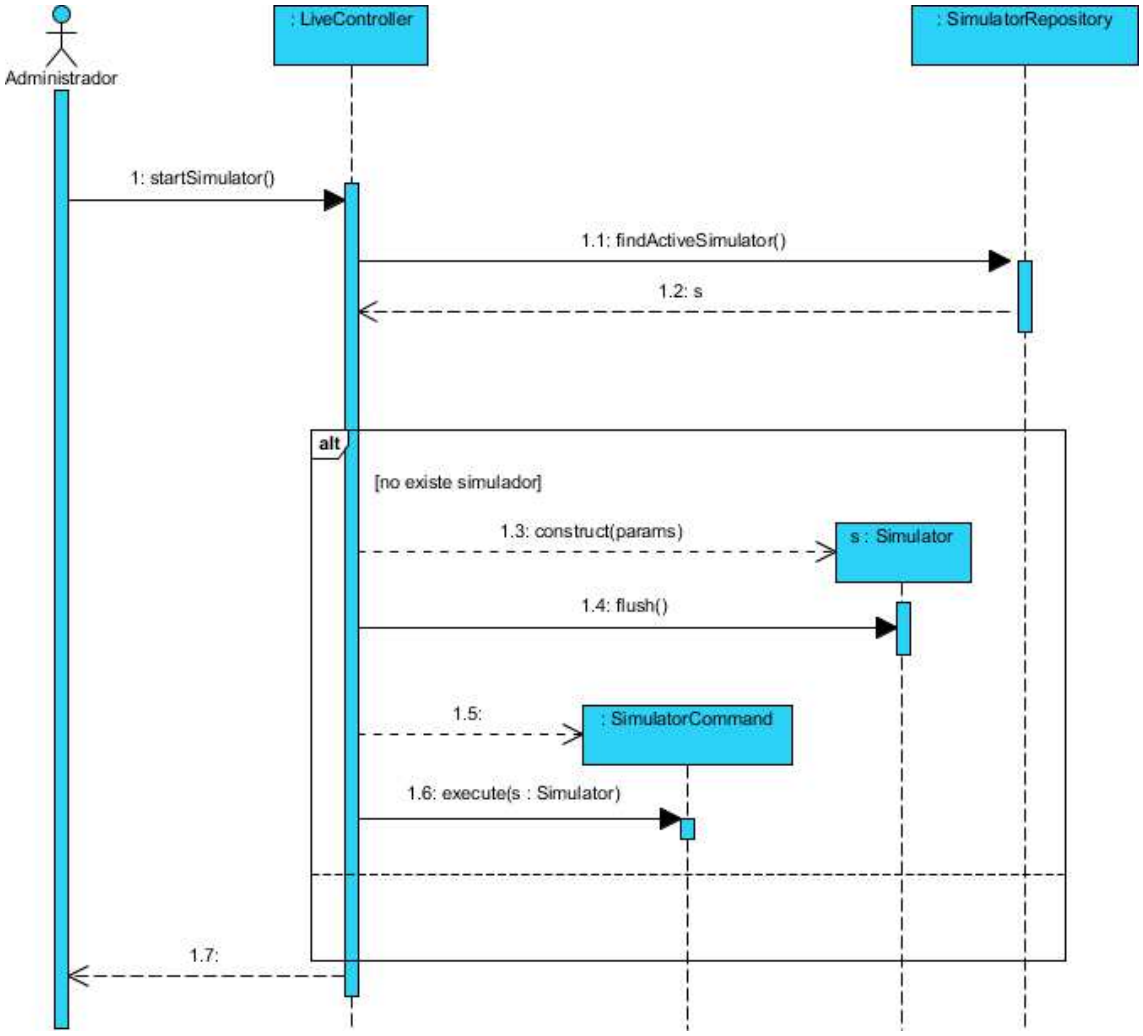
Desde la aplicación web se puede consultar el estado en tiempo real exactamente igual que de si un cliente móvil se tratara.

[12] Modelo-Vista-Controlado



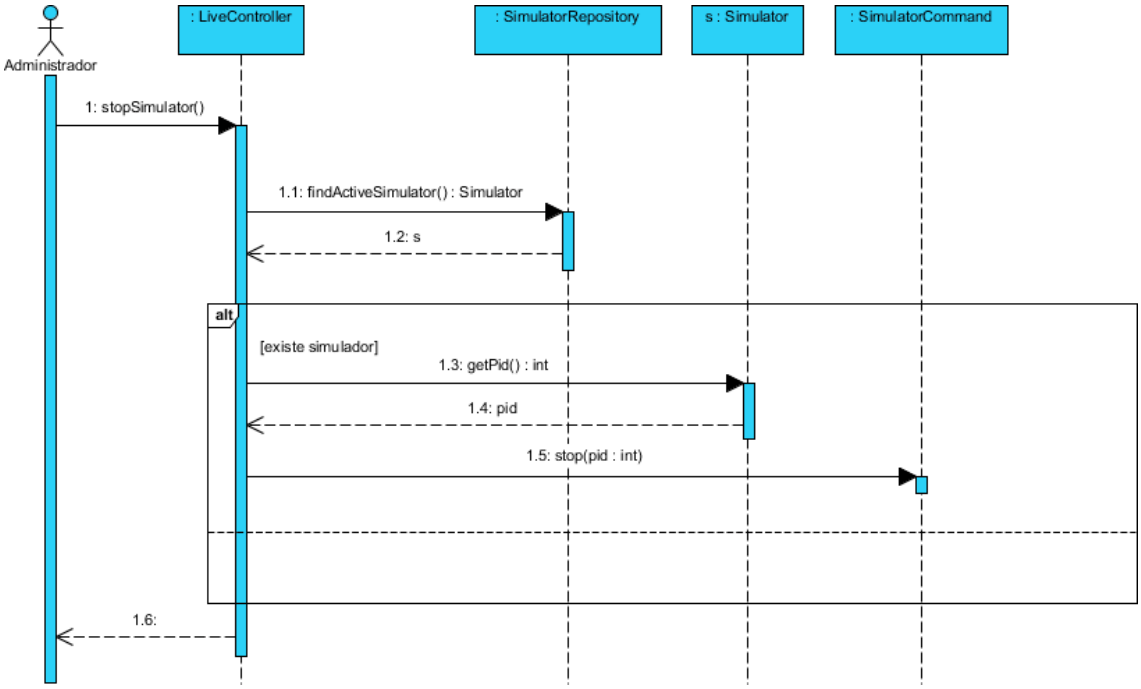
7.4.1.2. iniciarSimulador

Desde la aplicación web, los administradores pueden iniciar el proceso de simulación. Para ello deben proveer los parámetros necesarios para la simulación y una vez iniciado el simulador, la aplicación web se encarga de gestionar que sólo se pueda estar un simulador en funcionamiento en cualquier momento.



7.4.1.3. pararSimulador

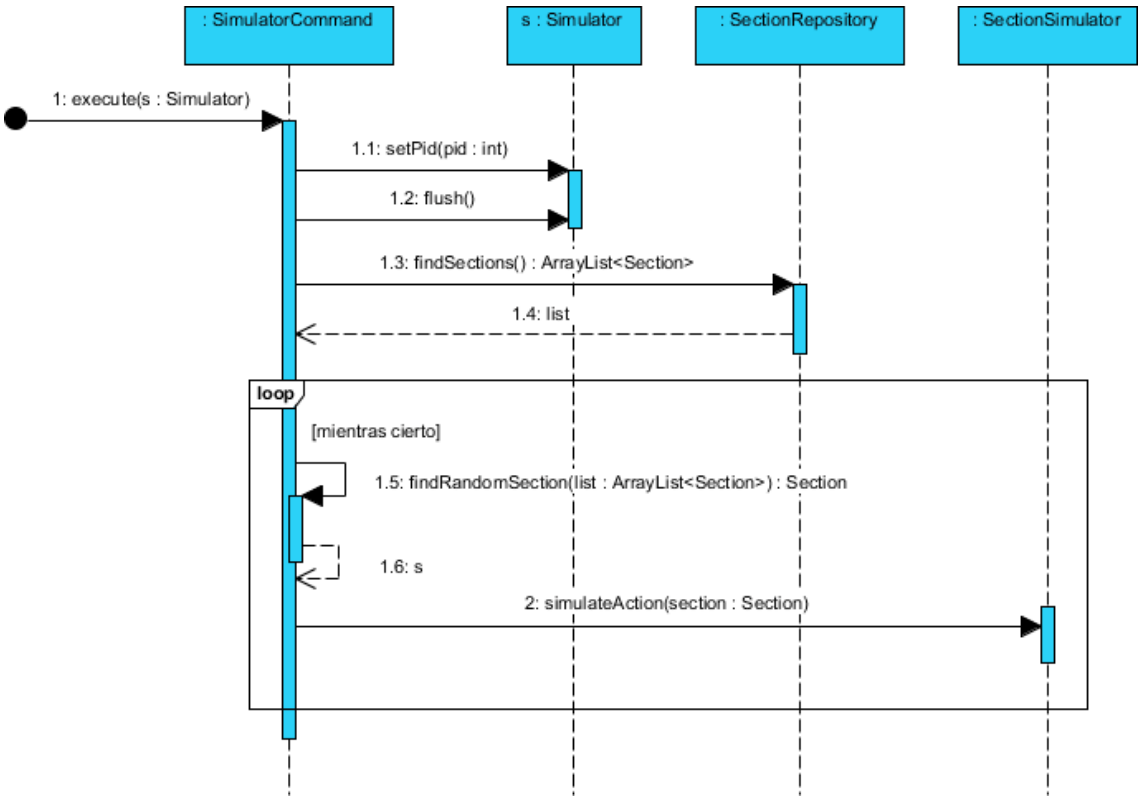
Desde la aplicación web, en caso de haber un simulador en ejecución, los administradores pueden solicitar su parada en cualquier momento.



7.4.2. Simulador

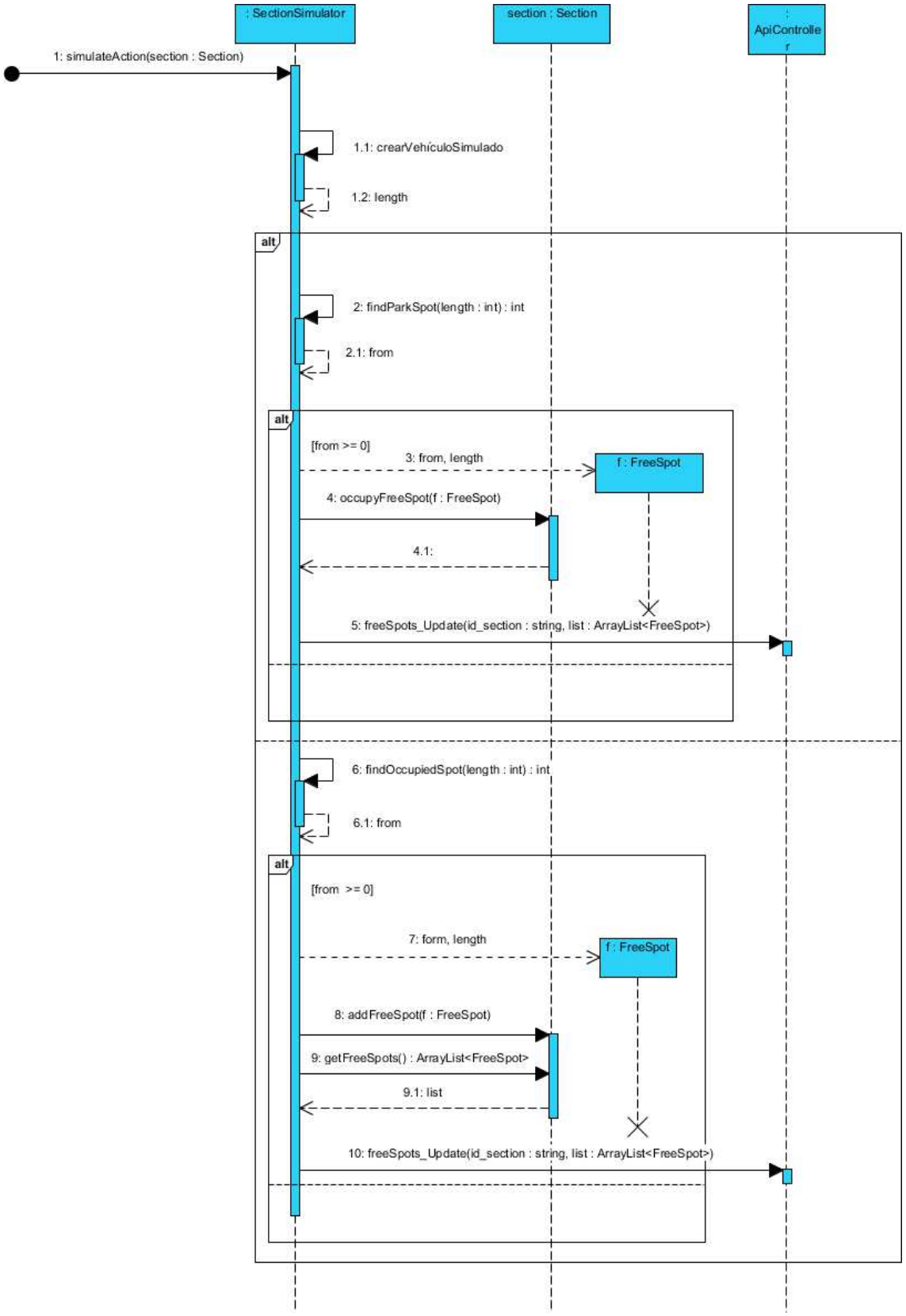
7.4.2.1. iniciarSimulador

Al iniciar el simulador se obtienen todas las secciones del sistema del rango a simular. Una vez obtenidas mediante un bucle infinito se selecciona una sección de forma aleatoria y se simula una acción en dicha sección.



La simulación de una acción en una sección consiste en:

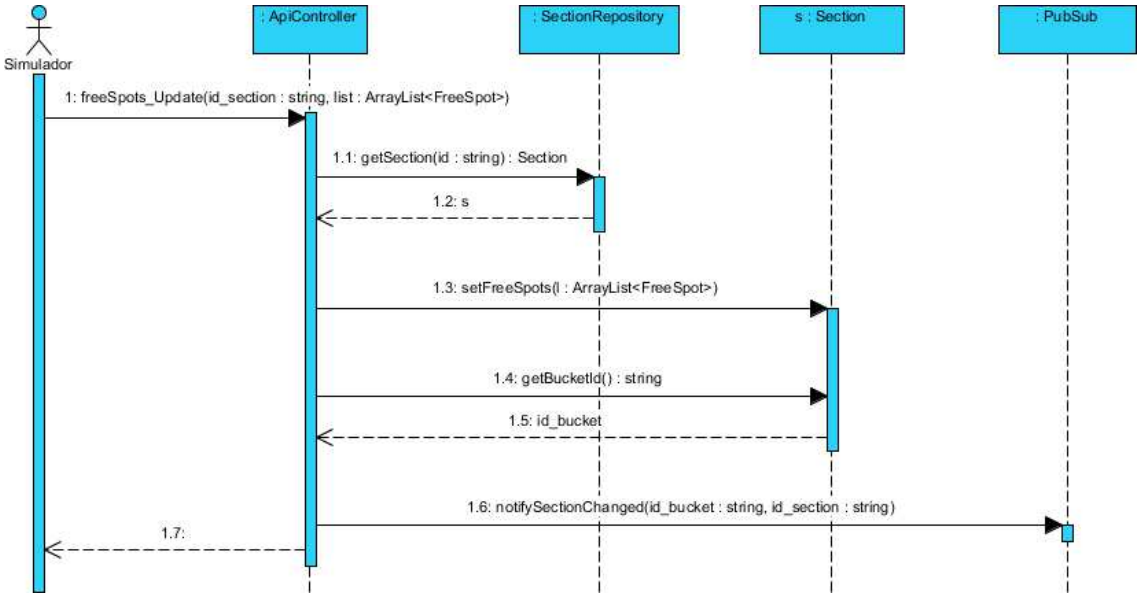
1. Elegir si la acción a realizar será de aparcar o liberar plaza de aparcamiento.
2. Simular el tamaño de un vehículo.
3. Encontrar la posición en la que el vehículo puede realizar la acción.
4. Simular la acción y guardar los cambios mediante la Api como haría un sensor.


: SectionSimulator
section : Section
: ApiControlle r
1: simulateAction(section : Section)
1.1: crearVehículoSimulado
1.2: length
alt
2: findParkSpot(length : int) : int
2.1: from
alt
[from >= 0]
3: from, length
f : FreeSpot
4: occupyFreeSpot(f : FreeSpot)
4.1:
5: freeSpots_Update(id_section : string, list : ArrayList<FreeSpot>)
6: findOccupiedSpot(length : int) : int
6.1: from
alt
[from >= 0]
7: form, length
f : FreeSpot
8: addFreeSpot(f : FreeSpot)
9: getFreeSpots() : ArrayList<FreeSpot>
9.1: list
10: freeSpots_Update(id_section : string, list : ArrayList<FreeSpot>)

7.4.3. API

7.4.3.1. actualizarPlazasLibres

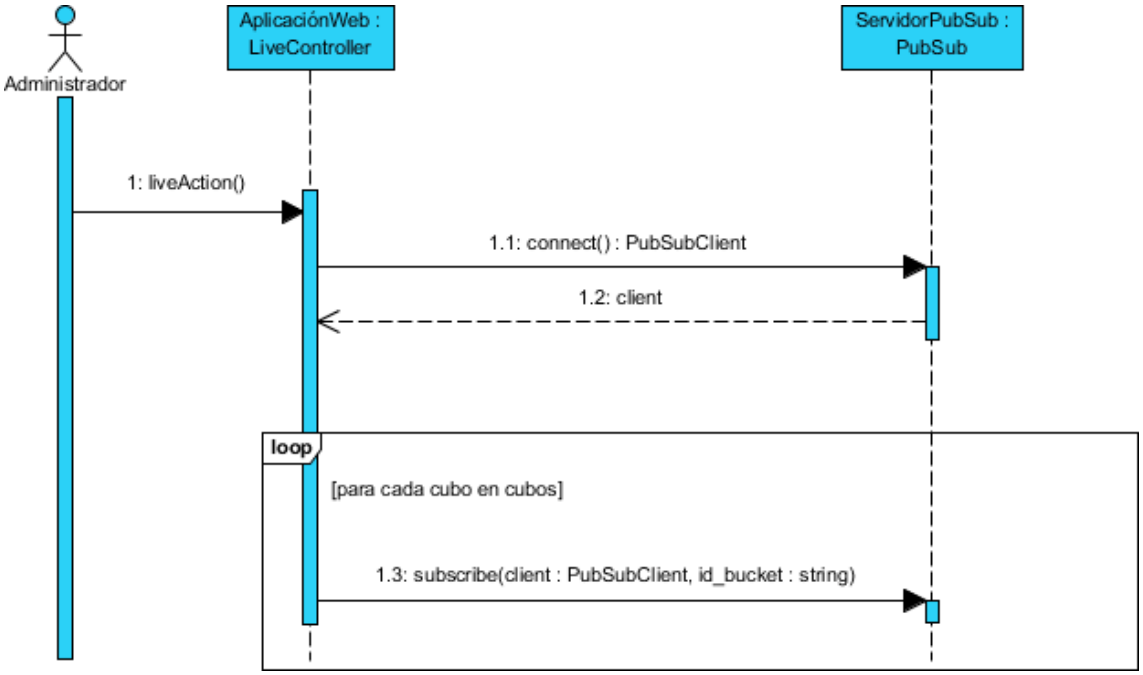
Cuando la API recibe la notificación de actualizar las plazas libres de una sección debe por una parte actualizar la sección en la base de datos, y por otra, notificar a todos los clientes de la actualización. Para realizar la notificación se notifica al servidor PubSub la sección que ha sido modificado y éste se encarga de notificar a los clientes afectados.



7.4.4. Servidor PubSub

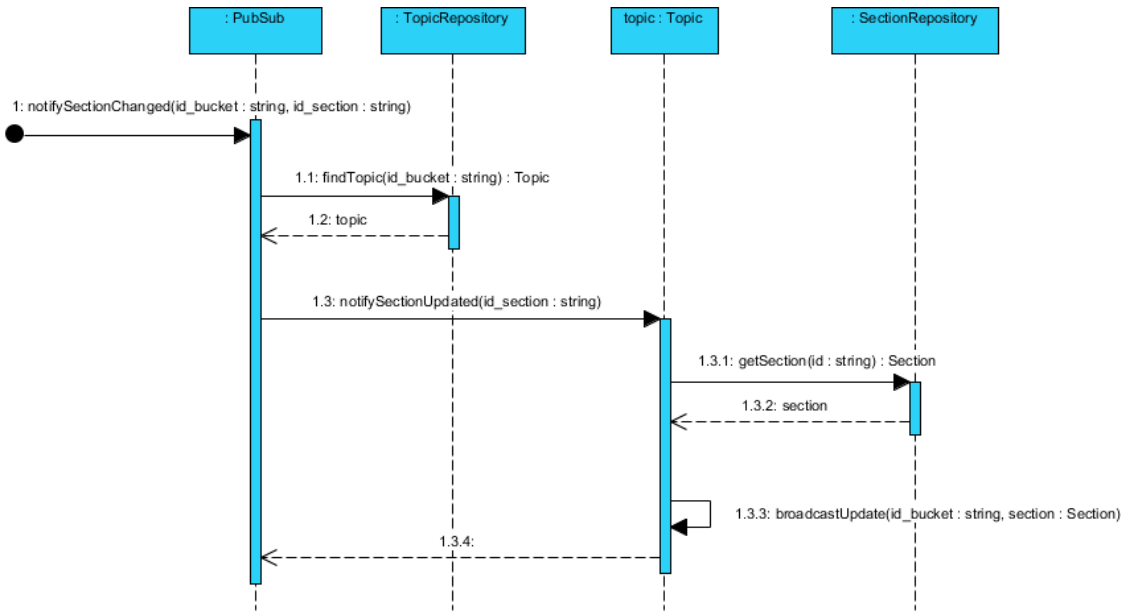
7.4.4.1. suscribirseCubo

Cuando un cliente solicita una suscripción a un cubo, debe recibir, si el cubo existe, toda la información actualizada de las secciones que pertenecen al cubo y las notificaciones de los cambios que puedan aparecer en las secciones que pertenecen al cubo al que se ha suscrito el cliente.



7.4.4.2. notificarSecciónActualizada

Cuando el servidor PubSub recibe una notificación de notificar una actualización de una sección, debe encontrar el tema correspondiente a la sección basándose en el cubo al que pertenece la sección, y notificar a todos los suscriptores los datos actualizados de la sección.



8. IMPLEMENTACIÓN

Como se ha descrito en el apartado de diseño, se tienen que implementar 5 componentes independientes que deben comunicarse entre sí.

Para el desarrollo se ha divido el proyecto en diversas fases, cada una de ellas permitirá abordar la siguiente fase con una base sólida. Estas fases son:

1. Implementación del sistema de gestión.
2. Implementación del simulador.
3. Implementación del servidor PubSub
4. Implementación de consulta en tiempo real.
5. Implementación de aplicación móvil.

A continuación comentaré cada fase en que ha consistido y los temas a destacar de cada una de ellas.

8.1. IMPLEMENTACIÓN DEL SISTEMA DE GESTIÓN

El primer punto a desarrollar ha sido el sistema de gestión. Antes de empezar cualquier otro componente, es necesario disponer de un sistema que permita gestionar la base de datos de secciones y proveer la base sobre la que se construirán todos los componentes.

8.1.1. Symfony, Doctrine y las dependencias.

El primer paso ha sido crear el entorno de trabajo. En un principio se planteó el uso el *Symfony3*, pero durante el desarrollo se tuvo que bajar la versión a *Symfony2.8* por problemas con dependencias.

Las dependencias han sido en algunos momentos un problema serio ya que algunos componentes necesitaban versiones diferentes de las mismas dependencias. En concreto el mayor problema ha sido la dependencia de *Doctrine*.

Doctrine es un *ORM*[13] par *PHP* muy usado en *Symfony*. Para la aplicación era necesario el componente de *Doctrine* para *MongoDB*, para así poder utilizar el propio doctrine tanto para *MariaDB* como para *MongoDB*.

[13] https://es.wikipedia.org/wiki/Mapeo_objeto-relacional

El problema del componente de *Doctrine* para *MongoDB* es que utiliza la librería de *PHP-Mongo*[14], la cual es una librería obsoleta y solo disponible hasta PHP 5.6. Ubuntu 16.04 instala por defecto PHP7 con lo cual era necesario instalar el aplicativo en Ubuntu 14.04 o forzar la instalación de PHP5.6 en Ubuntu 16.04. El problema de instalar una versión diferente a la oficial es que es muy fácil que en una actualización de algún paquete se instale la nueva versión de PHP si no se especifica activamente la versión del paquete a instalar, por lo que hay que ir con cuidado.

Resuelto el problema de la versión de PHP e instalada la versión obsoleta de la librería Mongo para PHP nos encontramos con otro problema, durante el desarrollo se necesitaba el uso de CustomCollections[15] de Doctrine, esta funcionalidad fue introducida en la versión 1.1 pero el paquete Doctrine-ODM no la tenía implementada y se encontraba en una rama de desarrollo. Por suerte después de comunicarlo a la comunidad[16] encargada del desarrollo de Doctrine-ODM, actualizaron el paquete rápidamente para poder hacer uso de esta funcionalidad.

8.1.2. La base de datos

Al principio del proyecto, las secciones iban a ser líneas rectas o curvas, donde las líneas curvas serían arcos definidos por un radio un punto de inicio y un punto final.

Se comenzó a desarrollar así el sistema de gestión, pero rápidamente nos dimos cuenta que aunque a nivel técnico los arcos cumplían su función, en el uso diario la gestión de secciones de esta manera era impracticable. Por un lado la definición del arco era una tarea tediosa, y por otro no se adaptaba bien a todas las calles.

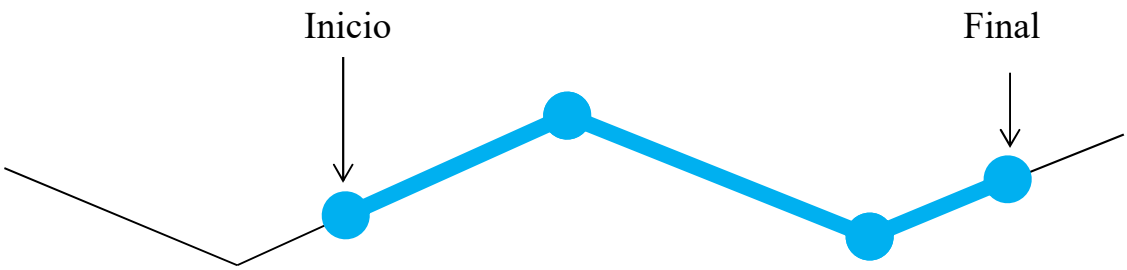
Este problema se resolvió sustituyendo la forma en que las secciones eran definidas, así pasaron de ser líneas rectas y arcos a ser poli-líneas. El uso de poli-líneas permite a las secciones adaptarse a las calles que representan sea cual sea su forma.

Las poli-líneas sin embargo añaden una complejidad a la aplicación, que se manifiesta en el cálculo de restricciones y plazas libres. Estas restricciones y plazas libres están definidas por una distancia desde el inicio de la sección y una longitud. En el caso de una línea recta o un arco es fácil de ubicar esta restricción con el uso de geometría, pero en el caso de una poli-

[14] http://php.net/manual/es/book.mongo.php
[15] http://docs.doctrine-project.org/projects/doctrine-mongodb-odm/en/latest/reference/custom-collections.html
[16] https://github.com/doctrine/mongodb-odm/issues/1415

línea es necesario implementar un algoritmo específico para definir esta restricción. Por lo tanto una restricción pasa de ser un segmento o arco a ser también otra poli-línea.

El algoritmo para calcular la poli-línea resultante de una restricción consiste en iterar los segmentos de la poli-línea para obtener el punto exacto de inicio y final, para posteriormente construir una poli-línea desde estos puntos calcando el recorrido de la poli-línea original.



La forma en que usuario introduce las secciones se implementó con la ayuda de la API de *Google Maps*[17] para *Javascript*, permitiendo dibujar las secciones sobre un mapa visual, agilizando así el proceso y evitando errores en caso de introducir directamente posiciones geográficas.

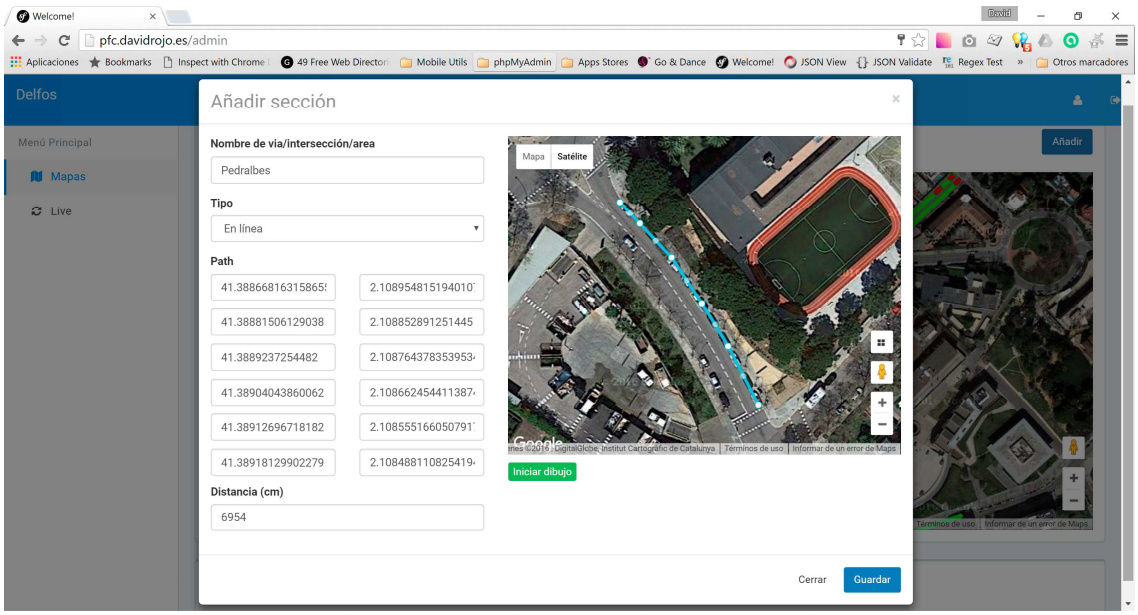


Ilustración 1 Formulario para añadir una restricción al sistema

En la consulta de secciones introducidas también se hace uso de Google Maps, permitiendo al usuario mover el mapa y automáticamente mostrar las secciones de la zona que se está visualizando y el listado con los detalles de estas secciones.

[17] https://developers.google.com/maps/documentation/javascript/?hl=es

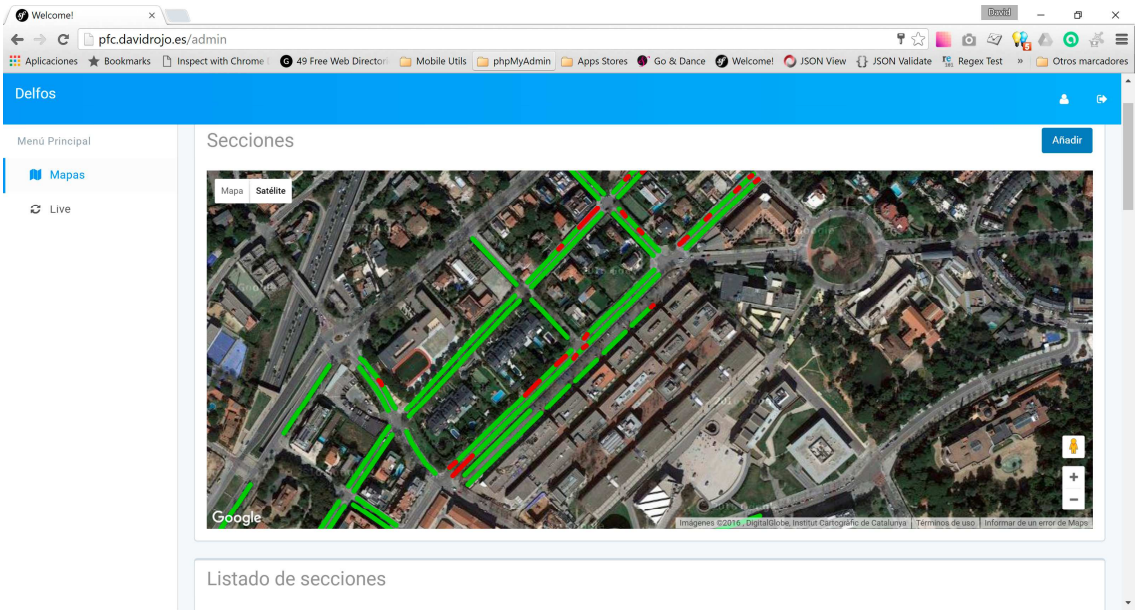


Ilustración 2 Listado de secciones introducidas con restricciones

A la hora de introducir las restricciones el usuario también visualiza la información que está introduciendo actualizándose el mapa automáticamente, así el usuario ve exactamente el resultado final de sus datos introducidos.

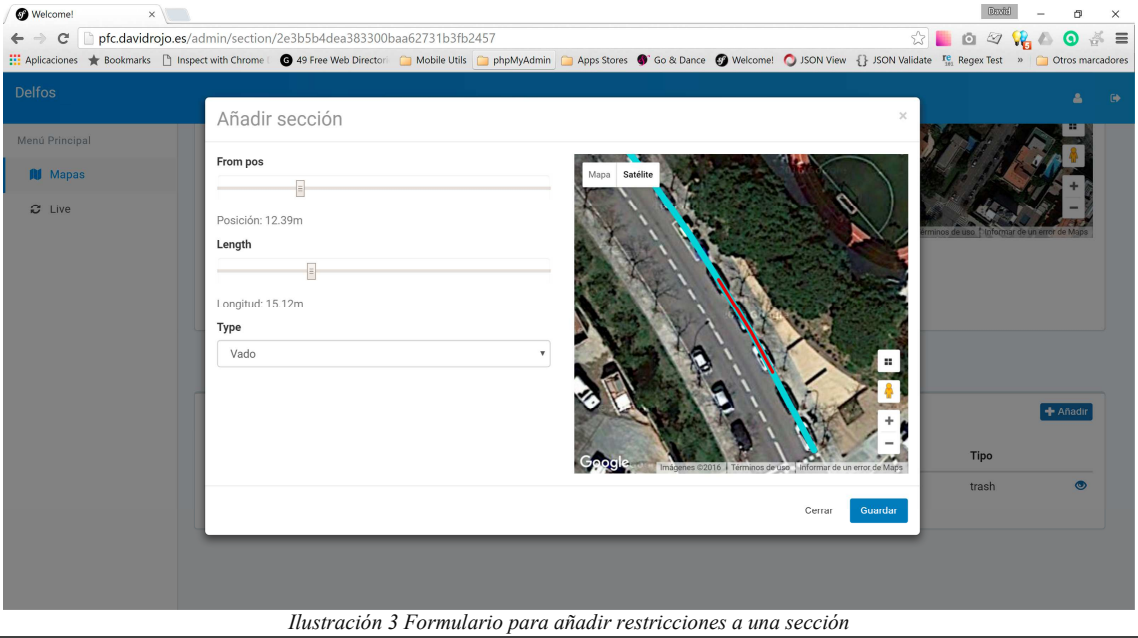


Ilustración 3 Formulario para añadir restricciones a una sección

Con el sistema de gestión de secciones finalizado el siguiente paso fue nutrir de información al sistema, añadiendo secciones de una zona concreta y comprobando que sus funcionalidades desarrolladas cubren las necesidades.

Con el sistema nutrido de datos el siguiente paso sería simular la actividad sobre la base de datos.

8.2.IMPLEMENTACIÓN DEL SIMULADOR Y LA API

El simulador es un sustituto del sistema de detección de plazas real, su finalidad es permitirnos utilizar el sistema como si de verdad estuviesen vehículos circulando.

En el proyecto se ha desarrollado un simulador de forma sencilla ya que no es uno de los objetivos del proyecto, y por lo tanto no es necesario que sea muy sofisticado.

Al simulador se le han desarrollado varias formas de funcionar según los parámetros recibidos:

- Simular una sección
- Simular una serie de cubos
- Simular una región geográfica

Independientemente de los parámetros recibidos el funcionamiento es el mismo, obtener todas las secciones de la zona a simular y ejecutar aleatoriamente acciones sobre estas secciones.

Durante el desarrollo del simulador se permitió la detección de bugs tanto del sistema como del propio simulador Al ser un componente que genera información aleatoria dentro de unos parámetros permitía detectar errores como si de una prueba en el mundo real se tratase.

Al simulador se le implementó un sistema para que mostrase las acciones que estaba ejecutando y así observar la actividad generada.

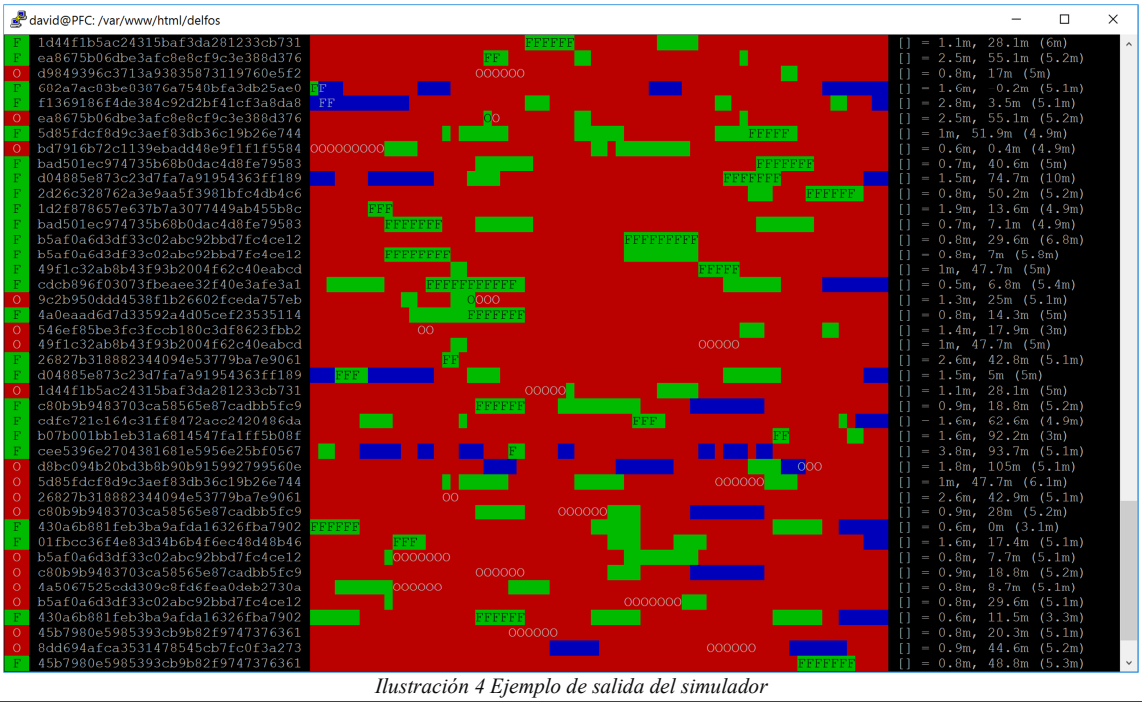


Ilustración 4 Ejemplo de salida del simulador

Por cada acción realizada el simulador muestra:

- Acción realizada (F = liberar, O = ocupar).
- Id(entificador) de la sección
- Bloques de estado
  - Fondo rojo = ocupado
  - Fondo verde = Libre
  - Fondo azul = Restricción
  - F = Zona liberada
  - O = Zona ocupada
- Longitud de cada bloque del estado
- Inicio de la acción
- Longitud de la acción

Como la salida por pantalla en modo texto es muy limitada, las secciones se muestran ocupando todas el mismo ancho de pantalla, pero como cada sección tiene una longitud diferente, la longitud de cada bloque de estado es el ancho en metros de cada carácter de la sección dibujada,

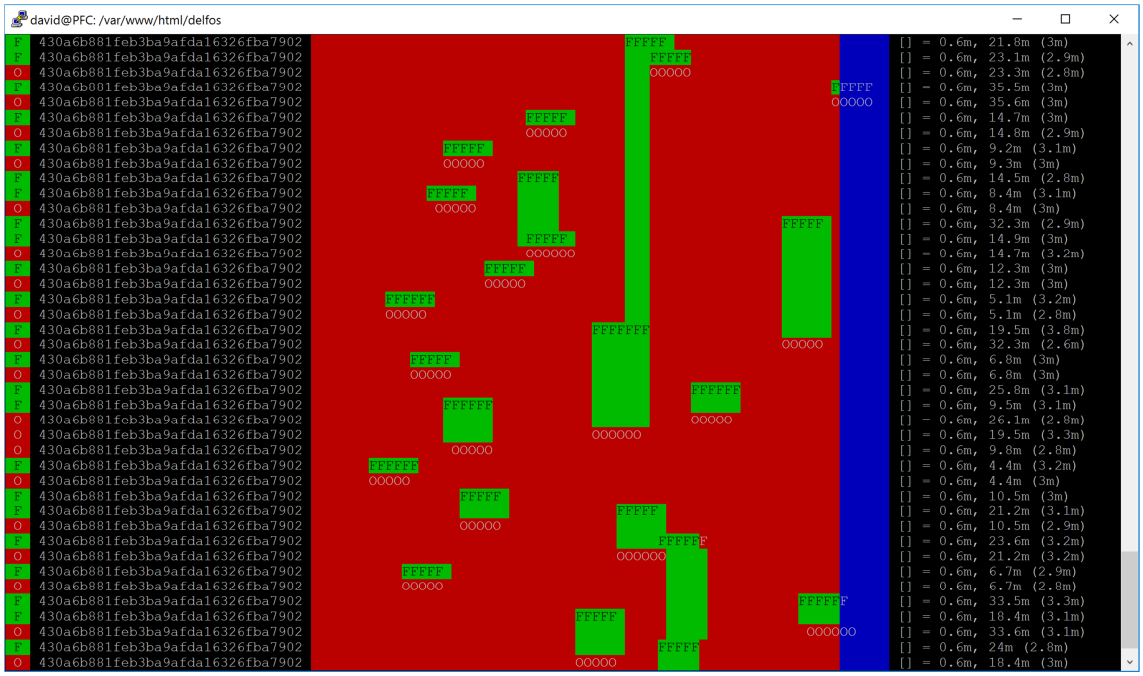


Ilustración 5 Ejemplo de actividad sobre una única sección

Al simulador se le permite simular la liberación y ocupación de plazas en restricciones. Ya que en la vida real, las plazas con restricciones también son utilizadas por los vehículos habilitados.

Aunque al simulador se le permite en la implementación acceder a la base de datos directamente para consultar información, a la hora de guardar los cambios debía hacer uso de la API, ya que, como indica la especificación, es como se comunicarían los sensores con el sistema y por lo tanto el simulador debía cumplir con el flujo de comunicaciones del sensor. Por este motivo en esta fase del proyecto se implementó la API.

La API se implementa como una API Restful. Después de investigar se decidió no utilizar ningún componente nuevo en Symfony para este menester, ya que la API a desarrollar no es de gran complejidad. Básicamente con el sistema de enrutamiento de Symfony y retornando respuestas en JSON podíamos cumplir con las necesidades.

Un tema que sí que era importante en la implementación de la API era el definir el método de acceso a cada ruta, GET, POST, PUT o DELETE. En nuestra API solo necesitábamos métodos GET para consultar y POST para actualizar datos.

Para implementar el tipo de acceso en cada ruta mediante Symfony sólo es necesario añadir la anotación a cada método.

```
/**
 * @Route("/api/sections/{idSection}/free-spots", name="api_section_update")
 * @Method({"POST"})
 */
public function freeSpots_Update(Request $request, $idSection){

}
```

Ilustración 6 Ejemplo de definición de método y ruta

8.3.IMPLEMENTACIÓN DEL SERVIDOR PUBSUB

La implementación del servidor PubSub es una tarea que requirió mucha investigación y poca programación.

Para la implementación del servidor PubSub se utilizó el componente WebSocketsBundle[18] de GeniusesOfSymfony. Este componente se encarga de toda la gestión de temas y de comunicaciones, haciendo que solo tengamos que programar que información se tiene que enviar en cada momento y a quien se tiene que enviar.

[18] https://github.com/GeniusesOfSymfony/WebSocketBundle

8.3.1. Implementación de temas

Para implementar un tema simplemente es necesario crear una clase que implemente *TopicInterface* y definir las rutas de suscripción en un fichero *routing*. Esta clase será la encargada de gestionar las suscripciones y las comunicaciones.

El método *onSubscribe* se ejecutará cuando un cliente se suscriba a un tema, recibiendo como parámetros los definidos en el fichero de *routing* y el objeto *Topic* que representa el tema al que el cliente quiere suscribirse.

Mediante el objeto *Topic* podemos enviar mensajes a todos los suscriptores ejecutando la función *broadcast*. Esta función permite añadir filtros para definir los usuarios que queremos que reciban los mensajes o los usuarios que queremos que no reciban los mensajes.

En nuestro caso, cuando un usuario se suscribe a un tema, es necesario enviarle toda la información de las secciones del cubo asociado, pero sólo a éste usuario, el resto de suscriptores no deben recibir nada.

```php
    /**
     * This will receive any Subscription requests for this topic.
     *
     * @param ConnectionInterface $connection
     * @param Topic $topic
     * @param WampRequest $request
     * @return void
     */
    public function onSubscribe(ConnectionInterface $connection, Topic $topic, WampRequest $request){
        $attr = $request->getAttributes();
        $bucketId = $attr->get("bucketId");

        $bucket = $this->em->getRepository('DelfosBundle:SectionBucket')->findOneById($bucketId);
        if ($bucket){
            $this->em->getManager()->detach($bucket);
            $response = [];
            foreach($bucket->getSections() as $section){
                $this->em->getManager()->detach($section);
                $c = $section->toJson();
                $response[] = $c;
            }

            $topic->broadcast([
                'action' => 'initialize',
                'bucket_id' => $bucketId,
                'sections' => $response
            ], [], [$connection->WAMP->sessionId]);
            echo "Client subscribed to ".$bucketId.' sent '.count($response).' sections in bucket'.PHP_EOL;
        }
        else{
            echo "No bucket found with id " . $bucketId.PHP_EOL;
        }
    }
```

Ilustración 7 Método onSubscribe del tema Bucket

El método *onUnSubscribe* se ejecutará cuando un cliente se dé de baja de un tema, en nuestro caso no debemos realizar ninguna acción.

El método *onPublish* se ejecutará cuando un cliente publique algún mensaje en el tema, en nuestro caso los clientes no pueden publicar temas por lo que no debemos realizar ninguna acción.

8.3.2. Pusher

Para permitir a la API publicar mensajes debemos implementar la interfaz *PushableTopicInterface* en nuestro tema e implementar la función *onPush*. También hemos de configurar el método que utilizaremos para realizar esta acción que puede ser *ZeroMQ*[19], *AMQP* [20]o WebSockets. En nuestro caso utilizamos la librería ZeroMQ.

El funcionamiento de la función *onPush* es similar a *onPublish*. Recibimos como parámetros el Topic y los datos a publicar y mediante la función broadcast transmitimos esta información a todos los suscriptores del tema.

```
89      /**
90       * @param Topic        $topic
91       * @param WampRequest  $request
92       * @param array|string $data
93       * @param string       $provider The name of pusher who push the data
94       */
95      public function onPush(Topic $topic, WampRequest $request, $data, $provider)
96      {
97          if ($data['action'] == 'update'){
98              $section = $this->em->getRepository('DelfosBundle:Section')->findOneById($data['sectionId']);
99              $this->em->getManager()->detach($section);
100             if ($section){
101                 $topic->broadcast([
102                     'action' =>  'update',
103                     'bucket_id' => $section->getBucket()->getId(),
104                     'sections' => [$section->toJson()]
105                 ]);
106                 echo "Sending message to update section " . $section->getId().PHP_EOL;
107             }
108             else{
109                 echo "Push section not found ".$data['sectionId'];
110             }
111         }
112     }
```

Ilustración 8 Método onPush del tema Bucket

8.3.3. Configuración

La configuración consiste en 3 archivos:

- **Services.yml**: Definimos un nuevo servicio con nuestra clase BucketTopic y argumentos y le asignamos el nombre topic_bucket para utilizar en el resto de configuraciones.

```
6   services:
7       delfos.topic_bucket:
8           class: DelfosBundle\Topic\BucketTopic
9           arguments:
10            entityManager: "@doctrine_mongodb"
```

- **Routing.yml**: Definimos las rutas de cada tema.

[19] http://zeromq.org/
[20] https://www.amqp.org/

```yaml
topic_bucket:
    channel: delfos/buckets/{bucketId}
    handler:
        callback: 'delfos.bucket'
    requirements:
        bucketId:
            pattern: "[0-9]+"
```

- **Config.yml**: Definimos la configuración general del servidor PubSub, Pushers y temas incluidos.

```yaml
gos_web_socket:
    server:
        port: 8080              #The port the socket server will listen on
        host: 192.168.18.128    #The host ip to bind to
        router:
            resources:
                - @DelfosBundle/Resources/config/pubsub/routing.yml
    topics:
        - @delfos.topic_bucket
    pushers:
        zmq:
            default: true
            host: 127.0.0.1
            port: 5555
            persistent: true
            protocol: tcp
```

El servidor PubSub se configura para que escuche en el puerto 8080 de la IP pública del servidor. El servidor para recibir notificaciones push se configura para que escuche en el puerto 5555 de la IP local del servidor.

Una vez configurado simplemente tenemos que ejecutar el servidor PubSub y veremos cómo atiende a peticiones remotas y locales.

```
david@PFC:/var/www/html/delfos$ php bin/console gos:websocket:server
[2016-09-06 13:40:45] websocket.INFO: Starting web socket
[2016-09-06 13:40:45] websocket.INFO: ZMQ transport listening on 127.0.0.1:5555
[2016-09-06 13:40:45] websocket.INFO: Launching Ratchet on 192.168.18.128:8080 PID: 2562
```

Ilustración 9 Muestra de servidor PubSub en ejecución

8.3.4. Modificación de la API

Con el servidor PubSub implementado y funcionando sólo nos queda modificar la API para que cuando reciba una petición que altere las plazas libres, comunique a nuestro servidor PubSub mediante Push la sección que se ha modificado. Simplemente con obtener el servicio pusher podemos realizar la acción

```php
    public function notifySectionChanged(Section $section){
        $pusher = $this->container->get('gos_web_socket.zmq.pusher');
        $pusher->push(
            [
                'action' => 'update',
                'sectionId' => $section->getId()
            ],
            'topic_bucket',
            [
                'bucketId' => $section->getBucket()->getId()
            ]);
    }
```

8.3.5. Doctrine y su sistema de caché

En el código mostrado se puede ver que en algunos puntos se ejecuta la función *detach* de doctrine.

```
26      public function onSubscribe(ConnectionInterface $connection, Topic $topic, WampRequest $request){
27          $attr = $request->getAttributes();
28          $bucketId = $attr->get("bucketId");
29
30          $bucket = $this->em->getRepository('DelfosBundle:SectionBucket')->findOneById($bucketId);
31          if ($bucket){
32              $this->em->getManager()->detach($bucket);
33              $response = [];
34              foreach($bucket->getSections() as $section){
```

Ilustración 11 Ejemplo de detach

Symfony es un framework para aplicaciones web, y doctrine es un ORM para gestionar la base de datos. Doctrine implementa un sistema de caché, el cual permite que si se consulta dos veces el mismo elemento, doctrine ejecute una única consulta a la base de datos. Esta operativa es correcta para su uso en un entorno web, ya que el tiempo de ejecución es muy corto, permitiendo ahorrar mucho tiempo a la aplicación por consultas duplicadas, pero en nuestro caso, al ser una aplicación que se ejecutará de forma indefinida no nos interesa este comportamiento.

En el siguiente gráfico podemos observar el funcionamiento de la caché de doctrine y como los cambios aplicados por la Api no se ven reflejados en las consultas del cliente.

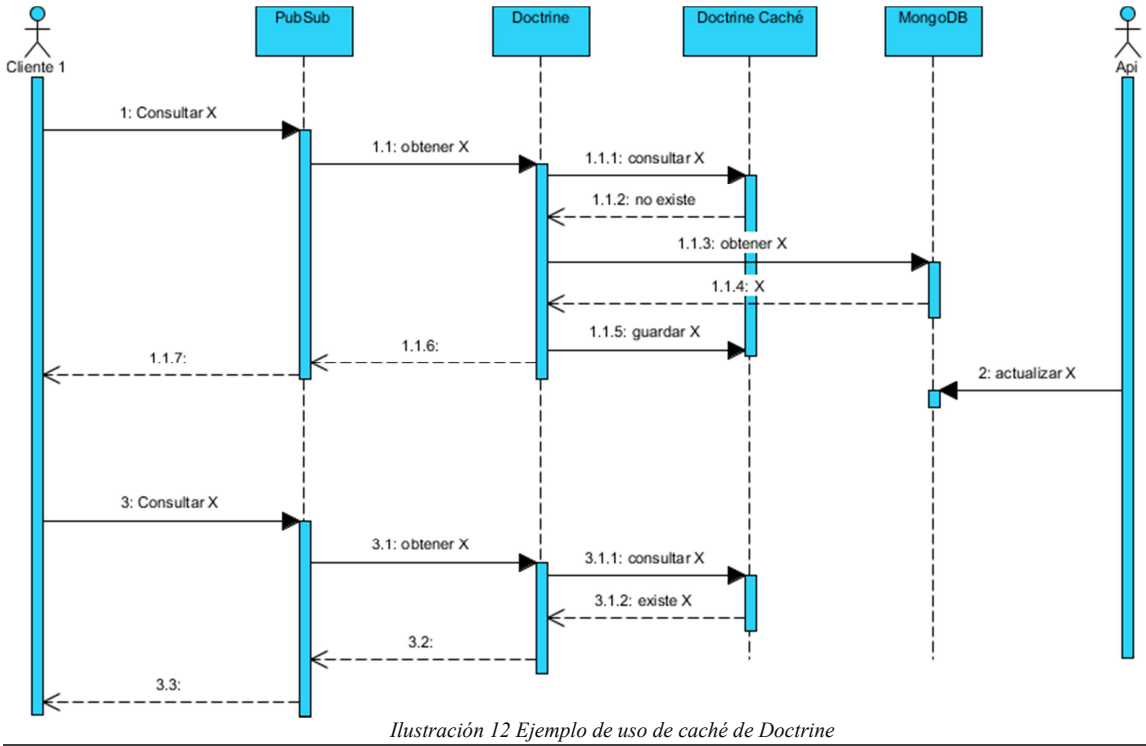


Ilustración 12 Ejemplo de uso de caché de Doctrine

Por ello, en nuestro servidor PubSub, es necesario anular activamente la caché cada vez que hagamos una consulta. El procedimiento consiste en después de realizar la consulta ejecutar la función *detach* del *Entity Manager* sobre el objeto resultante de la consulta, esto forzará a doctrine a obtener de nuevo el objeto actualizado del origen de datos.

8.4.IMPLEMENTACIÓN DE CONSULTA EN TIEMPO REAL

El siguiente paso del desarrollo era comprobar que todos los componentes funcionan correctamente y solventar los errores que puedan surgir en un entorno amigable y fácil de depurar.

Para esto se desarrolló en el panel de administración un cliente para el servidor *PubSub* en *javascript* que mostrase el estado de las plazas libres para una región y así comprobar el sistema de suscripción y baja de *buckets*.

También se añadió unos botones para poder iniciar y parar el simulador de forma rápida desde la interfaz web.

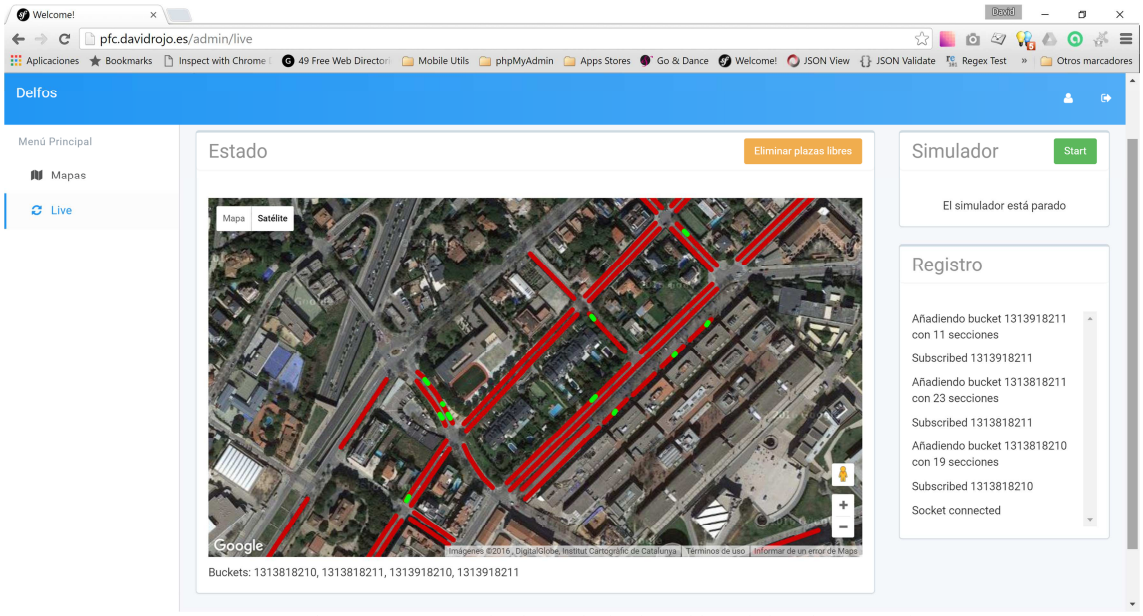


Ilustración 13 Consulta de estado de plazas libres de la aplicación web

Adicionalmente se añadió un botón para eliminar las plazas libres y resetear el sistema fácilmente.

Para la conexión desde *Javascript* al servidor *PubSub* se utilizan las librerías incluidas en el paquete *WebSocketsBundle* basadas en *Autobahn JS*[21].

[21] http://autobahn.ws/js/

La aplicación automáticamente inicia la suscripción a *buckets* a medida que se mueve el mapa, realizando la suscripción a los *buckets* que aparecen en el mapa y sus adyacentes, garantizando así que se obtendrán todas las secciones visibles.

A la hora de suscribirse a un *bucket*, la suscripción puede ser satisfactoria o no, ya que es posible que solicitemos la suscripción a un *bucket* y éste no exista. Además debemos mantener un control de los *buckets* en los que estamos suscritos para evitar suscribirnos varias veces y para poder darnos de baja cuando no sea necesario seguir recibiendo actualizaciones

Para ello definiremos una variable *listSubscribe* la cual contendrá la lista de buckets en la que estamos suscritos y solo se actualizará cuando hayamos recibido la confirmación de la suscripción.

Para realizar la conexión al servidor PubSub utilizaremos la función *connect* de *WS*. Acto seguido definimos los *callbacks* que se ejecutaran cuando la conexión se establezca o se desconecte.

```
226             // Connect to pubsub
227             var socketSession =  false;
228             var websocket = WS.connect("ws://{{ pubsub_host }}:{{ pubsub_port }}");
229             websocket.on("socket/connect", function(session){
230                 toastr.success("Socket conneted");
231                 socketSession = session;
232             });
233
234             websocket.on("socket/disconnect", function(error){
235                 toastr.warning("Socket disconnected" + error.reason);
236             });
```

Ilustración 14 Código Javascript para conectar a servidor PubSub

Si la conexión del *WebSocket* se pierde, la librería *WS* realizará automáticamente 5 intentos de reconexión para recuperar la conexión, en caso de fallar los 5 intentos desconectará completamente la conexión.

En el momento que sea necesario suscribirse a un nuevo bucket ejecutaremos la función *suscribe* del objeto *Websocket* que anteriormente habíamos obtenido tras la conexión.

```
238         function subscribe(bucketId){
239             if  (socketSession){
240                 socketSession.subscribe("delfos/buckets/" + bucketId, function(uri, payload){
241                     if (payload.action == "initialize"){
242                         $('#log-socket').prepend('<p>' + "Subscribed " + bucketId + '</p>');
243                         listSubscribe.push(payload.bucket_id);
244                         initializeMapBucket(payload.bucket_id, payload.sections);
245                     }
246                     else if (payload.action == "update"){
247                         updateSection(payload.bucket_id, payload.sections);
248                     }
249                 });
250             }
251         }
```

Ilustración 15 Código Javascript para suscribirse a un tema

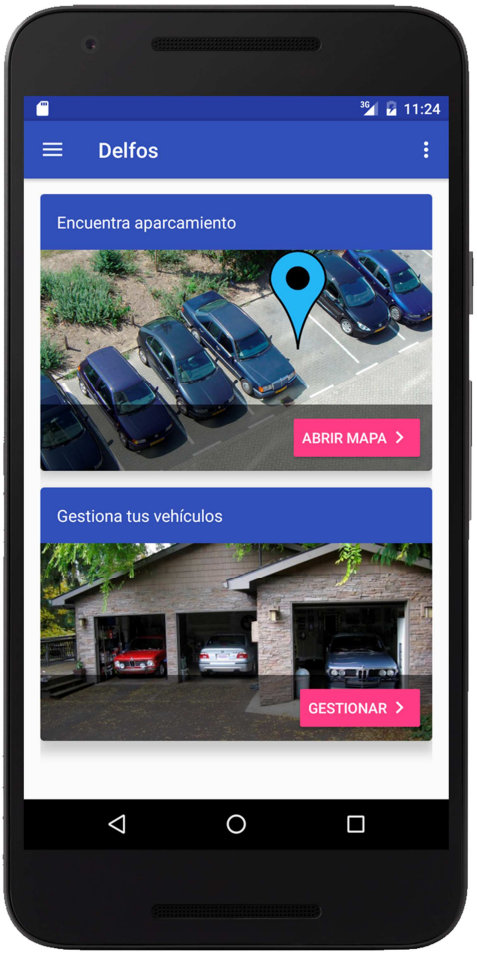
A la función *suscribe* pasamos como parámetro la ruta del tema al que queremos suscribirnos y el *callback* que se ejecutará cada vez que se reciba un evento. En el *callback* recibiremos la ruta del tema que ha provocado el evento y el *payload* que contiene un *JSON* con el contenido del evento enviado.

En nuestro caso disponemos de dos eventos, cuando nos suscribimos y cuando recibimos una actualización. La distinción del evento se realiza mediante un parámetro enviado por el servidor PubSub en la variable *action.*

Toda la información recibida de la notificación se recibe en el parámetro *payload* en la función *callback.*

8.5.IMPLEMENTACIÓN DE LA APLICACIÓN MÓVIL

Una vez tenemos todo el sistema funcionando y comprobado que el sistema de suscripciones funciona, podemos pasar a la implementación de la aplicación móvil que será la que utilizaran los clientes finales.



La aplicación móvil permite gestionar la flota de vehículos del usuario y consultar las plazas libres disponibles para su flota.

Para implementar la aplicación se ha utilizado Android Studio como IDE de desarrollo que incorpora diversas utilidades para facilitar y agilizar el trabajo de desarrollo.

8.5.1. Gestión de vehículos

Desde la aplicación web, los administradores pueden ver todas las plazas libres de aparcamiento, independientemente de su tamaño y sus restricciones

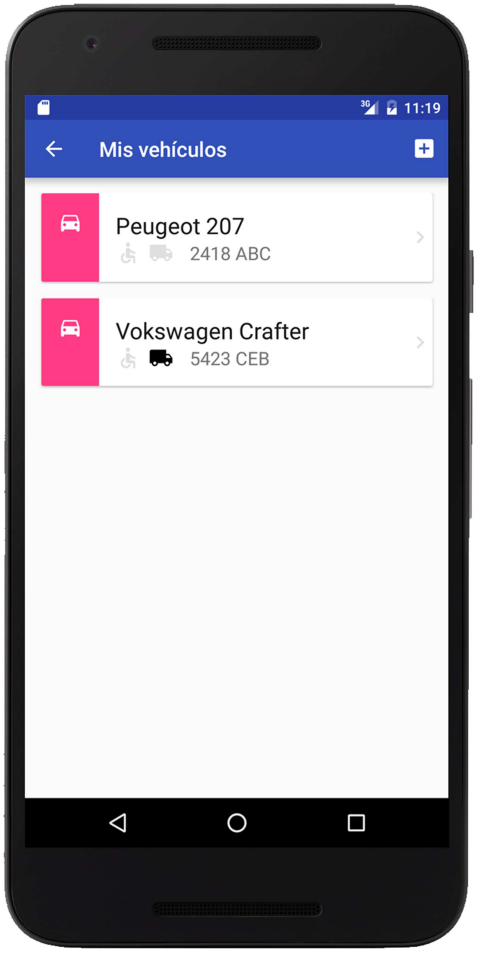
de uso, en cambio, el cliente debe ver sólo aquellas que se ajusten a su vehículo. Por este motivo, desde la sección de gestión de vehículos, el cliente podrá dar de alta los vehículos que disponga y sus características, para que, posteriormente, pueda buscar plazas libres para el que esté conduciendo en cada momento.

En el listado de vehículos se mostrará la información más importante de cada vehículo del cliente, permitiendo así identificar rápidamente si estos vehículos están configurados correctamente.

Para guardar los datos de los vehículos creados por el usuario disponemos de diversos formas de almacenaje en los dispositivos Android.

- **Shared Preferences**: Permite guardar datos en formato clave-valor.
- **Internal Storage**: Permite guardar ficheros en la memoria interna del dispositivo.
- **External Storage**: Permite guardar ficheros en la memoria externa del dispositivo.
- **SQLite Database**: Permite guardar datos de forma estructurada en una base de datos privada.

Para nuestra aplicación la única forma que se ajusta a los requerimientos sería *SQLite Database*, pero sin embargo en este caso utilizaremos un sistema de base de datos llamado *Realm*[22] que actúa como *ORM* permitiendo guardar objetos java de forma rápida y sencilla.



Para mostrar el listado de vehículos se utiliza el componente *RecyclerView* de la *Support Library* de *Google*. Este componente permite mostrar listados ahorrando un gran uso de memoria a base de reutilizar componentes.

El componente *ListView* carga por cada elemento una vista en memoria que representa la visualización dentro del listado del elemento asociado, en cambio, el *RecyclerView* sólo carga

[22] https://realm.io/

las vistas necesarias para visualizar los elementos que el usuario puede ver en pantalla, y en caso que el usuario mueva la lista y aparezca un nuevo elemento aprovecha una de las vistas ya creadas y la actualiza con los valores del nuevo elemento a visualizar aprovechando alguna de las vistas que ya no aparecen en la pantalla.

Este comportamiento se gestiona creando un adaptador que extienda de *RecyclerView.Adapter* e implementando los métodos *onCreateViewHolder* y *onBindViewHolder*.

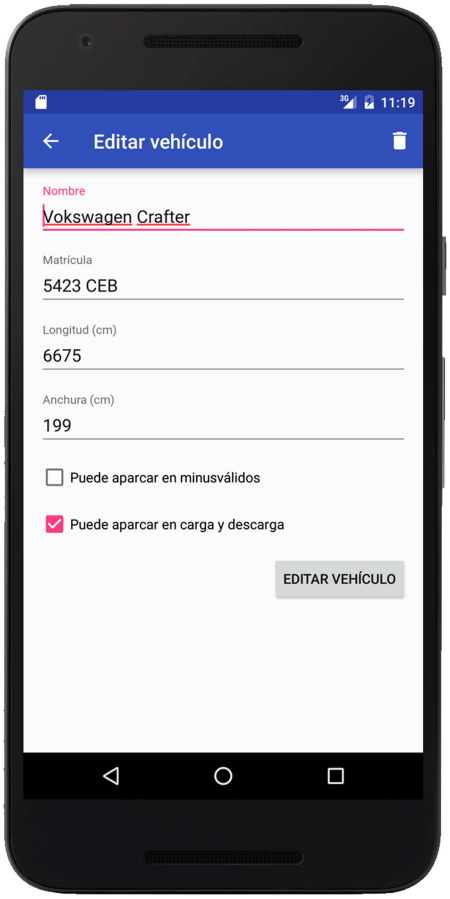
onCreateViewHolder: Éste método se ejecuta cada vez que el *RecyclerView* debe generar una nueva vista para un elemento. En éste método aún no se sabe que se va a mostrar, simplemente debemos crear un *ViewHolder* y configurar la vista para más adelante personalizar con el elemento a mostrar

onBindViewHolder: Éste método se ejecuta en el momento en que se vincula a un *ViewHolder* un elemento en concreto, recibiendo como parámetros el *ViewHolder* a utilizar y la posición del elemento que debemos mostrar. Por lo tanto debemos obtener la información del elemento basándonos en la posición recibida y modificar el *ViewHolder* para personalizar su contenido.

Para cada vehículo que el cliente quiera dar de alta deberá introducir:

- Nombre del vehículo.
- Matrícula del vehículo.
- Longitud.
- Anchura.
- Si puede aparcar en plaza de minusválidos
- Si puede aparcar en zonas de carga y descarga.

La longitud del vehículo será utilizada para calcular en que plazas puede aparcar el vehículo determinando si el tamaño de la plaza es suficientemente largo.



La anchura del vehículo se utilizará también para determinar si el vehículo cabe en una plaza en el caso que la forma de aparcar en una sección sea en diagonal o en batería.

8.5.2. Suscripción a PubSub

La consulta de plazas se realizará de la misma manera que se hizo en la aplicación web, mediante el servidor PubSub, pero con la diferencia que en este caso debemos filtrar las plazas libres según las restricciones del usuario.

Para poder realizar la conexión al servidor PubSub utilizaremos la librería *Autobahn Android*[23]. Debido a que esta librería se encuentra en *Github* pero no en los repositorios *Maven* necesitaremos añadir *Jitpack*[24] para importar el repositorio mediante *Gradle* añadiendo en nuestro fichero *gradle* del proyecto en la sección *repositories*:

```
maven { url "https://jitpack.io" }
```

Una vez hecho esto podemos incorporar al fichero *gradle* de nuestra aplicación el repositorio añadiendo dentro de la sección *dependencies*:

```
compile 'com.github.crossbario:autobahn-android:v0.5.2'
```

Debido a problemas a la hora de empaquetar el proyecto fue necesario excluir algunos ficheros, para ello añadimos en la sección *Android* del fichero *gradle* de nuestra aplicación:

```
packagingOptions {
    exclude "META-INF/INDEX.LIST"
    exclude "META-INF/ASL2.0"
}
```

La implementación del cliente para el servidor PubSub es muy similar a la realizada anteriormente en la aplicación Web. La idea consiste en el mismo principio, suscribirse a aquellos *buckets* que se muestren en el mapa y los buckets adyacentes y mantener la gestión de los *buckets* a los que estamos suscritos para en el momento de no visualizarlos darnos de baja.

Para ello debemos implementar la interfaz *Wamp.ConnectionHandler* en la clase que utilicemos para conectar con el servidor PubSub y conectar con nuestro servidor PubSub mediante un objeto *WampConnection* e implementar los métodos *onOpen* y *onClose* para controlar cuando la conexión se ha establecido o perdido.

[23] http://autobahn.ws/android/
[24] https://jitpack.io/

```
private final Wamp mConnection = new WampConnection();
private String wsuri = "";

wsuri = "ws://" + getResources().getString(R.string.ws_host) + ":" +
         getResources().getInteger(R.integer.ws_port);

mConnection.connect(wsuri, this);
```

Para suscribirnos a un *bucket*, al igual que en la aplicación Web, indicaremos la ruta de suscripción y el *callback* a ejecutar cuando recibamos una notificación.

```
private void subscribe(final int bucketId){
    if (mConnection !=  null && mConnection.isConnected()) {
        mConnection.subscribe(
            "delfos/buckets/" + bucketId,
            PubSubAction.class,
            new Wamp.EventHandler() {
              @Override
              public void onEvent(String topicUri, Object event) {
                // when we get an event, we safely can cast to
                // the type we specified previously
                try {
                    PubSubAction evt = (PubSubAction) event;

                    if (evt.getAction().equals("initialize")) {
                        // First connection to the socket
                        mapLines.addSections(evt.getSections());
                        mSubscriptions.add(new Integer(bucketId));
                    } else if (evt.getAction().equals("update")) {
                        // Topic publish a new update
                        for (Section s : evt.getSections()) {
                            mapLines.updateSection(s);
                        }
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
              }
        });
    }
}
```

En este caso debemos definir el tipo de objeto que recibiremos como *payload* de la notificación, en este caso un objeto *PubSubAction* y en la función de *callback* realizar la conversión del objeto.

8.5.3. Búsqueda de aparcamiento

La vista de búsqueda de plazas de aparcamiento muestra:

- Un mapa con la ubicación del vehículo.
- Las plazas libres en la región visualizada.
- Un botón para cambiar las preferencias.

- Un botón para cambiar la forma de visualizar las plazas libres.
- Un botón para centrar el mapa en la posición del vehículo.
- Un selector con los vehículos del usuario.

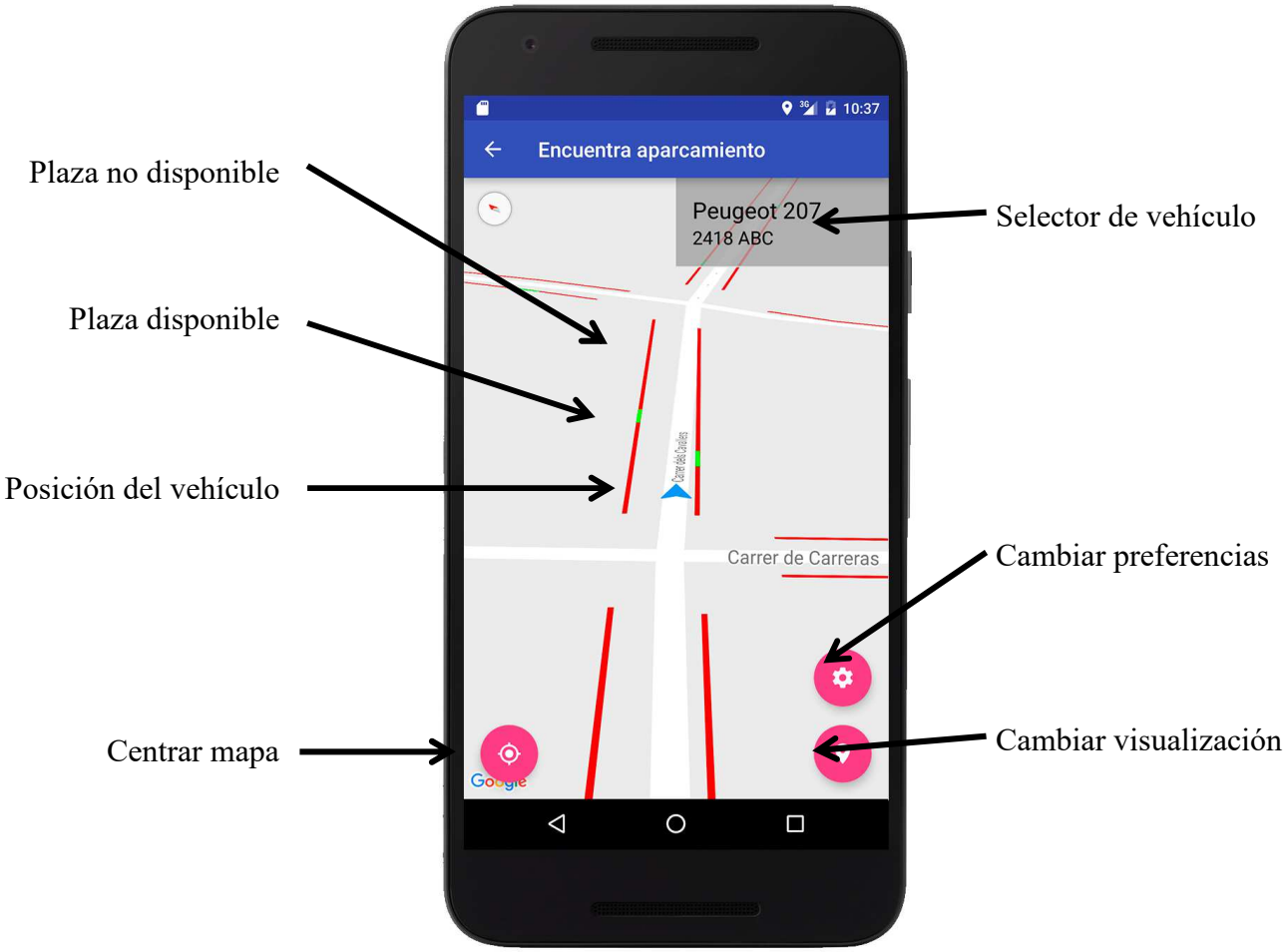


Ilustración 16 Pantalla de visualización de búsqueda de aparcamiento

La posición del vehículo se actualiza automáticamente a medida que el vehículo se mueve, siempre orientando la flecha en la dirección del movimiento y manteniendo la vista del mapa centrada en el vehículo.

El botón para centrar el mapa, únicamente se activa si se ha movido el mapa, de tal manera que al mover el mapa, éste deja de moverse junto al vehículo. Una vez centrado el mapa en el vehículo, éste vuelve a seguir al vehículo.

El cambio de visualización permite ver las plazas libres como puntos en lugar de líneas, esto permite una visualización más clara de los puntos dónde se puede aparcar.

El botón de cambiar preferencias permite establecer las preferencias de búsqueda de plazas.

El selector de vehículos permite cambiar el vehículo a otro vehículo del usuario, esto además cambia las preferencias en caso de búsqueda de plazas de minusválidos o carga y descarga, ya que estas propiedades dependen del vehículo. En este caso se notifica al usuario que se han cambiado las preferencias de visualización.

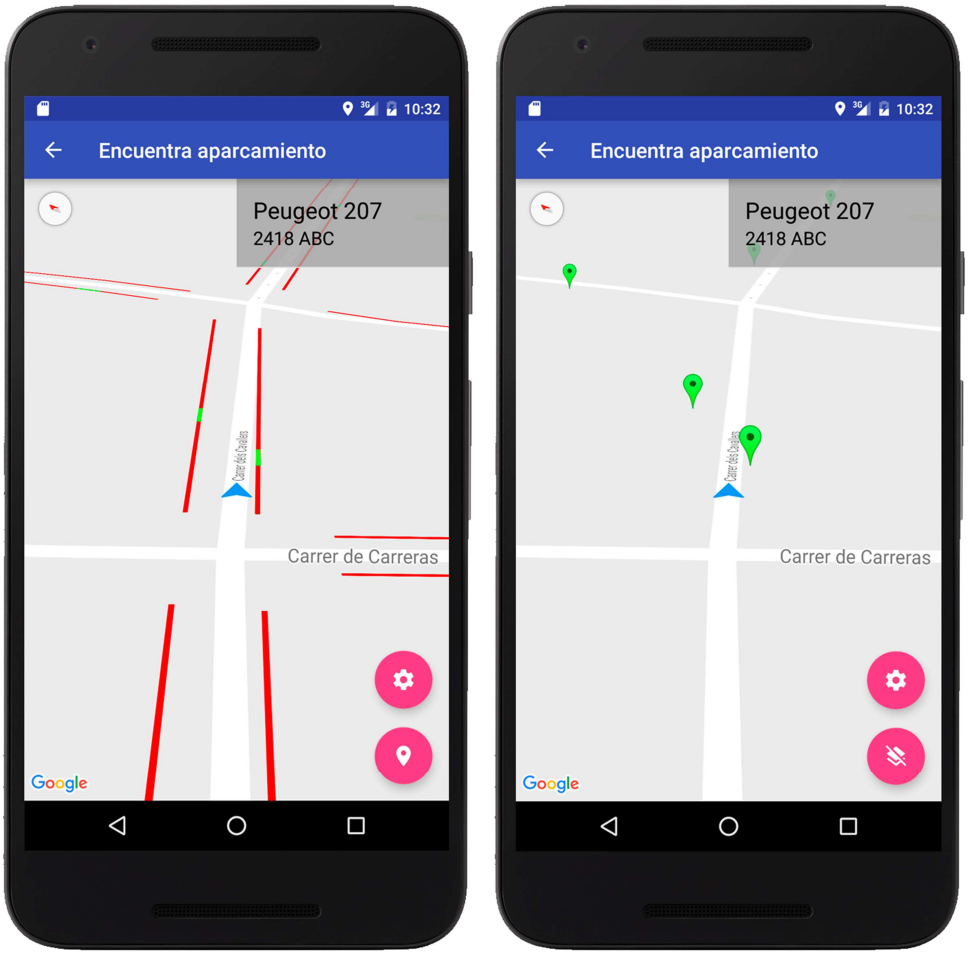


Ilustración 17 Muestra de mapa en formato de líneas y puntos

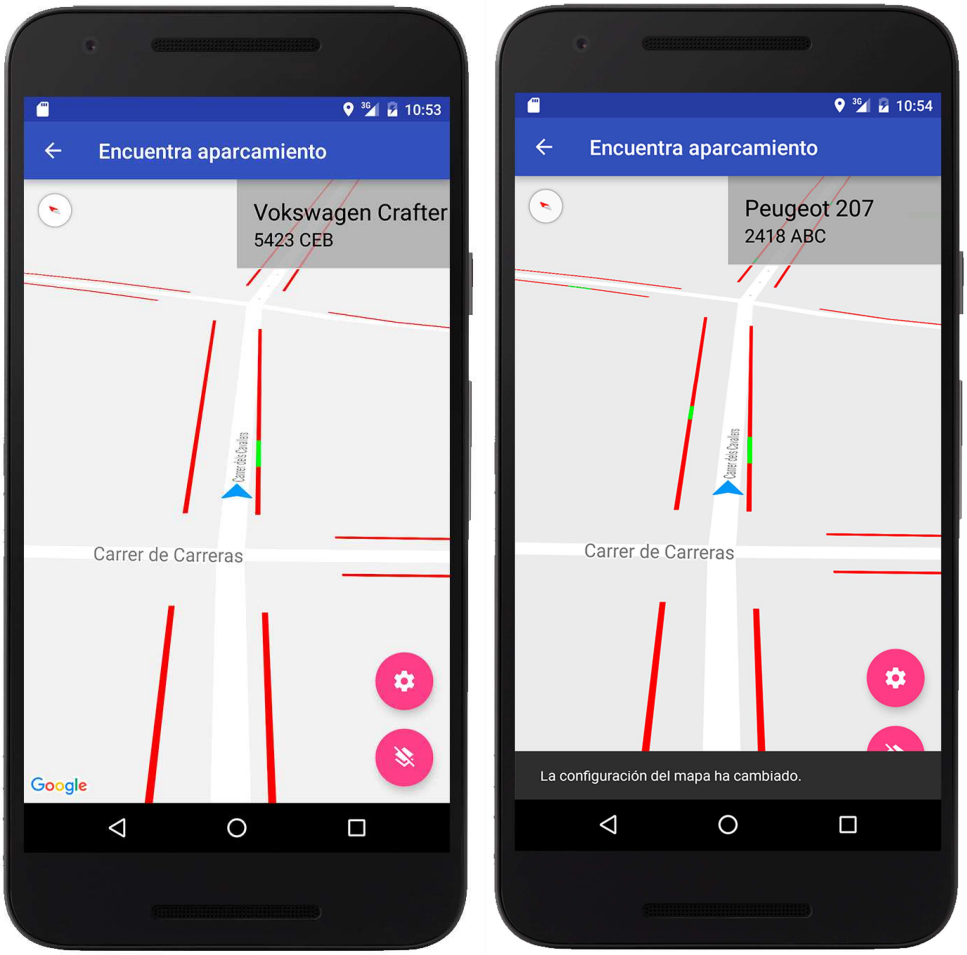


Ilustración 18 Ejemplo cambio de vehículo

Para mostrar los resultados en el mapa implementamos la clase *MapLines* que se encarga de gestionar como se muestran las plazas en el mapa. Esta clase recibe un objeto *ParkSearchSettings* que define las restricciones en la búsqueda de aparcamiento que consiste en el vehículo para el cual buscamos plaza y las restricciones que podemos utilizar, así como el margen de seguridad que el usuario quiera dejar entre su vehículo y el resto de vehículos.

En la clase *MapLines* tenemos un *HashMap* con todas las secciones y otro *Hashmap* con todas las secciones por cada *bucket*, de tal manera que podamos acceder a todas las secciones de un *bucket* y poder eliminarlas cuando se realice una baja de una suscripción de forma rápida.

El algoritmo para obtener las plazas libres de una sección tiene dos etapas.

1- Obtener las plazas libres de restricciones
2- Mostrar solo aquellas de tamaño superior al del vehículo.

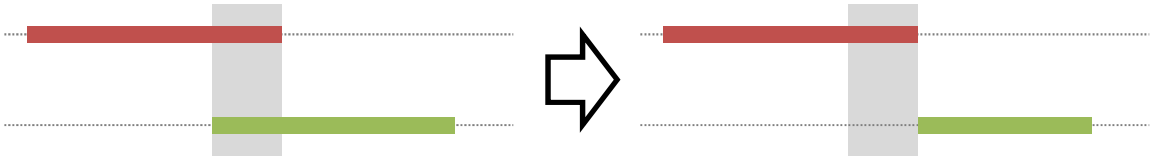
El algoritmo para determinar las plazas libres de restricciones consiste en una intersección de dos conjuntos de segmentos ordenados por su posición inicial en una dimensión.

El algoritmo consiste en iterar cada plaza libre y cada restricción de forma paralela, y en cada iteración comrpobar si la restricción es aplicable para el vehículo actual. Para ello se inicializa el algoritmo con un puntero a la primera plaza libre y otro a la primera restricción, y se finaliza cuando ambos punteros hayan finalizado sus respectivos arreglos, avanzando en cada iteración uno o ambos punteros. El coste temporal de este algoritmo es $O(n+m)$, siendo *n* el número de plazas y *m* el número de restricciones, resultando en un coste lineal.

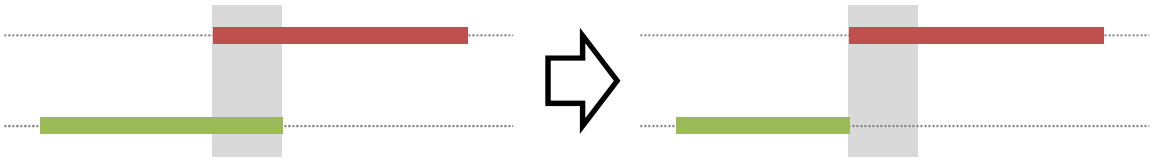
En cada iteración puede ocurrir una de las siguientes situaciones:

1- La restricción actual no afecta al vehículo: En ese caso adelantamos el puntero a la siguiente restricción.
2- La restricción actual impide al vehículo aparcar:
    a. La plaza libre y la restricción se solapan:
        i. La restricción solapa por la izquierda a la plaza libre: Modificamos la plaza libre para que empiece a partir del final de la restricción y avanzamos a la siguiente restricción.
        ii. La restricción solapa por la derecha a la plaza libre: Modificamos la plaza libre para que acabe al inicio de la restricción y avanzamos a la siguiente plaza libre.
        iii. La restricción es mayor que la plaza libre: No podemos aparcar, eliminamos la plaza libre pero no avanzamos ningún puntero, virtualmente el puntero de plaza libre apunta a la siguiente plaza libre.
        iv. La restricción se encuentra dentro de la plaza libre: Convertimos la plaza libre en dos plazas libres que excluyen la restricción, avanzamos a la siguiente restricción y continuamos desde la nueva plaza libre que ha quedado después de la restricción.
    b. La restricción actual acaba antes que la plaza libre actual: La restricción no es aplicable. Avanzamos el puntero a la siguiente restricción.
    c. La plaza libre actual acaba antes que la restricción actual: La plaza libre actual es apta para aparcar. Avanzamos el puntero a la siguiente plaza libre.

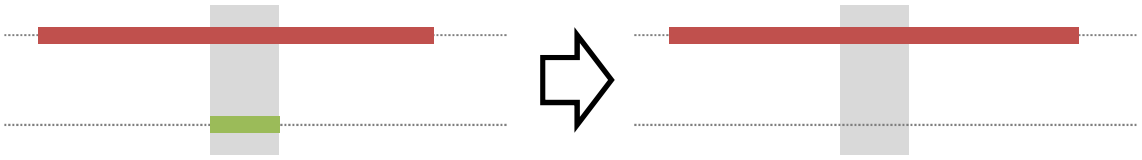
Caso i:



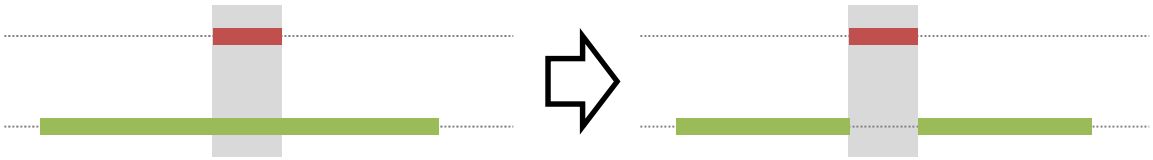
Casi ii:



Caso iii:



Caso iv:



Una vez obtenidas las plazas libres de restricciones a las que el cliente opta, filtrar aquellas suficientemente grandes para que el vehículo pueda aparcar es una tarea trivial, simplemente se debe iterar sobre las plazas y mostrar únicamente aquellas cuya longitud sea superior al tamaño del vehículo más el margen anterior y posterior de seguridad definido por el cliente.

El algoritmo completo en Java para obtener las plazas libres es el siguiente:

```
/**
 * Returns a list of available free spots after applying restrictions
 * @param restrictions
 * @return
 */
public  ArrayList<FreeSpot> getFreeSpotsFromRestrictions(
        ArrayList<Restriction> restrictions,
        ParkSearchSettings settings){

    int iRestriction = 0;
    int iSpot = 0;
```

```
// Create a copy of the spots
ArrayList<FreeSpot> freeSpots = new ArrayList<>();
for(Spot f : spots){
    freeSpots.add(new FreeSpot(f));
}

while(iSpot < freeSpots.size()
      && iRestriction < restrictions.size()){

    FreeSpot s = freeSpots.get(iSpot);
    Restriction r = restrictions.get(iRestriction);

    if (!settings.isAbleToPark(r)) {
        if (r.getEnd_pos() < s.getFrom_pos()){
            // Restriction ends before free spot,
            // advance restriction
            iRestriction++;
        }
        else if (s.getEnd_pos() < r.getFrom_pos()){
            // Spot ends before restriction, advance spot
            iSpot++;
        }
        else{
            // Overlapping
            if (r.getFrom_pos() <= s.getFrom_pos()) {
                if (r.getEnd_pos() < s.getEnd_pos()) {
                    // Restriction cuts the begining part of
                    // the free spot
                    s.setFrom_pos(r.getEnd_pos());
                    iRestriction++;
                } else {
                    // Restriction totally overlaps free spot
                    freeSpots.remove(iSpot);
                }
            } else {
                if (r.getEnd_pos() < s.getEnd_pos()){
                    // Restriction is inner the free spot,
                    // create two free spots
                    int endPos = s.getEnd_pos();
                    int length = s.getEnd_pos();

                    s.setLength(r.getFrom_pos()
                                - s.getFrom_pos());

                    FreeSpot n = new FreeSpot(s);
                    n.setFrom_pos(r.getEnd_pos());
                    n.setLength(endPos - r.getEnd_pos());
                    freeSpots.add(iSpot+1, n);

                    iRestriction++;
                    iSpot++;
                }
                else{
                    // Restriction cuts the end part of
                    // the free spot
                    s.setLength(r.getFrom_pos()
                                - s.getFrom_pos());

                    iSpot++;
                }
```

```
                    }
                }
            }
            else{
                iRestriction++;
            }
        }
        return freeSpots;
    }
```

Una vez obtenidas todas las plazas libres de una sección solo queda dibujarlas en el mapa. La plaza libre ahora mismo solo dispone de la información de punto de inicio y longitud, es necesario pasar esta información a un segmento de posiciones geográficas para dibujarla en el mapa. Esta conversión se realiza utilizando la ruta definida por la sección.

El objetivo de la función es encontrar en que segmentos se encuentra el punto inicial y final de la plaza libre, una vez encontrados, se calcula la posición exacta de dichos puntos dentro de cada segmento y se añaden todos los segmentos que se encuentren entre el punto inicial y final junto con la porción correspondiente del segmento inicial y final.

Para ahorrar cálculos, se han calculado con anterioridad las distancias de cada segmento de la ruta y se almacenan en el arreglo *distances* para poder consultar sus valores sin realizar de nuevo el cálculo cada vez.

La distancia entre dos puntos geográficos se realiza mediante la librería de Google Maps *SphericalUtil.* El cálculo es una aproximación, debido a que se utiliza un radio constante de la tierra para calcular las distancias. Si quisiéramos mayor precisión necesitaríamos conocer la elevación de cada punto.

A continuación se muestra la función en Java del cálculo de ruta de una restricción/plaza libre sobre una ruta:

```
/**
 * Returns a path resulting of extracting restriction from pos
 * and length from the path
 * provided
 * @param restriction
 * @param path
 * @return
 */
private List<LatLng> buildPath(
        Spot restriction,
        ArrayList<GeoPosition> path){

    ArrayList<LatLng> rpath = new ArrayList<>();
```

```
    // Distance between first point and start of current segment
    int lengthElapsed = 0;


    // Segment index for the first point
    int startIndex = 0;

    // Segment index for the last point
    int endIndex = 0;


    // Find start index
    while(startIndex < path.size() - 1 && restriction.getFrom_pos() >
lengthElapsed + this.distances[startIndex]){

        lengthElapsed += this.distances[startIndex];
        startIndex++;
    }

    if (startIndex >= path.size() - 1){
        Log.e("SpotsMap", "La posición inicial de la restricción "
                + restriction.getId()
                + " es superior al tamaño del trazado.");

        return  new ArrayList<LatLng>();
    }

    // Add first point position
    LatLng prevPoint = getAveragePoint(
        path.get(startIndex), path.get(startIndex+1),
       (restriction.getFrom_pos() - lengthElapsed) /
           this.distances[startIndex]);

    startLengthOffset = SphericalUtil.computeDistanceBetween(
         path.get(startIndex).toLatLng(),
         prevPoint) * 100;
    rpath.add(prevPoint);

    // Find end index
    endIndex = startIndex;
    while(endIndex < path.size() - 1 && restriction.getFrom_pos()+
restriction.getLength() > this.distances[endIndex] + lengthElapsed){

        lengthElapsed += this.distances[endIndex];
        endIndex++;
    }

    if (endIndex >= path.size() - 1){
        Log.e("SpotsMap", "La posición final de la restricción "
                + restriction.getId()
                + " es superior al tamaño del trazado.");
    }


    // Add next points that belong to path
    for(int i = startIndex+1; i <= endIndex; i++){
        rpath.add(path.get(i).toLatLng());
        prevPoint = path.get(i).toLatLng();
    }
```

```
    if (endIndex < path.size() - 1) {
        // add last point, if the algorithm has reachet the
        // whole array the last point is already added.
        // This could be due to misscalculations in distances
        // and precissions
        int distanceLeft = restriction.getLength()
             + restriction.getFrom_pos() - lengthElapsed;

        LatLng lastPoint = this.getAveragePoint(
            path.get(endIndex),
            path.get(endIndex + 1),
            distanceLeft / this.distances[endIndex]);

        rpath.add(lastPoint);
    }

    return rpath;
}
```

Una vez tenemos la ruta de cada plaza libre, solo resta mostrar esta información sobre el mapa. Para ello debemos construir una poli-línea utilizando *PolyLineOptions*.

```
List<LatLng> path = this.buildPath(r, section.getPath());

// Create polyline
PolylineOptions p = new PolylineOptions();
p.color(Color.GREEN);
p.width(13);
p.addAll(path);

// Add to map
Polyline cp = map.addPolyline(p);
```

También en nuestro caso añadimos un *marker* al mapa en el inicio de la plaza libre. El proceso es similar pero utilizando *MarkerOptions.* Aprovechamos también la poli-línea creada para obtener el *LatLng* inicial en el formato que necesita *MarkerOptions* y así ahorrarnos una conversión al ejecutar la función *position* de *MarkerOptions*. El marker utilizado será un *marker* por defecto de *Google Maps* de color verde.

```
BitmapDescriptor icon  = BitmapDescriptorFactory.defaultMarker(
        BitmapDescriptorFactory.HUE_GREEN);

MarkerOptions mo = new
MarkerOptions().position(p.getPoints().get(0)).icon(icon)
Marker m = map.addMarker(mo);
```

Por último, dependiendo del tipo de visualización a mostrar ocultaremos o mostraremos el marker o la poli-línea mediante la función *setVisible*.

También en este punto comprobamos si el tamaño de la plaza libre es superior a la del vehículo seleccionado.

```
if (settings.getMapStyle() == ParkSearchSettings.MAP_STYLE_LINES){
    m.setVisible(false);
    if (r.getLength() >= length) {
        cp.setVisible(true);
    }
}
elseif (
      settings.getMapStyle() == ParkSearchSettings.MAP_STYLE_MARKERS){

    if (r.getLength() >= length) {
        m.setVisible(true);
    }
    cp.setVisible(false);
}
else{
    m.setVisible(false);
    cp.setVisible(false);
}
```

9. PLAN ECONÓMICO

En esta sección s e realiza un plan de ejecución de tareas, asignando cada tarea a un rol y calculando el coste de desarrollo de cada rol.

9.1. PLANIFICACIÓN DE TAREAS DE IMPLEMENTACIÓN

Nombre de tarea	Duración	Nombres de los recursos
Aplicación web		
Análisis	30 horas	Analista
Diseño	30 horas	Diseñador
Gestión de usuarios		
Listado de usuarios	8 horas	Programador Senior web
Añadir usuario	6 horas	Programador Senior web
Editar usuario	4 horas	Programador Senior web
Gestión de secciones		
Listado de secciones	8 horas	Programador Senior web
Mapa listado de secciones	8 horas	Programador Senior web
Añadir sección	12 horas	Programador Senior web
Detalles de sección		
Editar sección	6 horas	Programador Senior web
Listado de restricciones	4 horas	Programador Senior web
Añadir restricción	12 horas	Programador Senior web
Editar restricción	6 horas	Programador Senior web
Estado del sistema	20 horas	Programador Senior web
Perfil usuario		
Cambiar datos personales	6 horas	Programador Senior web
Cambiar contraseña	3 horas	Programador Senior web
Servidor PubSub		
Análisis	35 horas	Analista

Suscripción de clientes	10 horas	Programador Senior web
Actualización de secciones	10 horas	Programador Senior web
Api		
Análisis	10 horas	Analista
Secciones		
Consultar secciones	8 horas	Programador Senior web
Consultar plazas libres	5 horas	Programador Senior web
Actualizar plazas libres	12 horas	Programador Senior web
Aplicación móvil		
Análisis	24 horas	Analista
Diseño	20 horas	Diseñador
Garaje		
Listado de vehículos	12 horas	Programador Android
Añadir vehículo	4 horas	Programador Android
Editar vehículo	4 horas	Programador Android
Eliminar vehículo	1 hora	Programador Android
Búsqueda de aparcamiento		
Consultar plazas	30 horas	Programador Android
Cambiar visualización	12 horas	Programador Android
Centrar mapa	4 horas	Programador Android
Cambiar vehículo	12 horas	Programador Android

A continuación se muestra la planificación temporal de la ejecución de la implementación del proyecto según las tareas y tiempos definidos, así como separando cada tarea por cada rol.

También se definen las restricciones de tareas que deben ejecutarse previamente para poder avanzar en cada una de ellas, permitiendo paralelizar el desarrollo para cada rol.

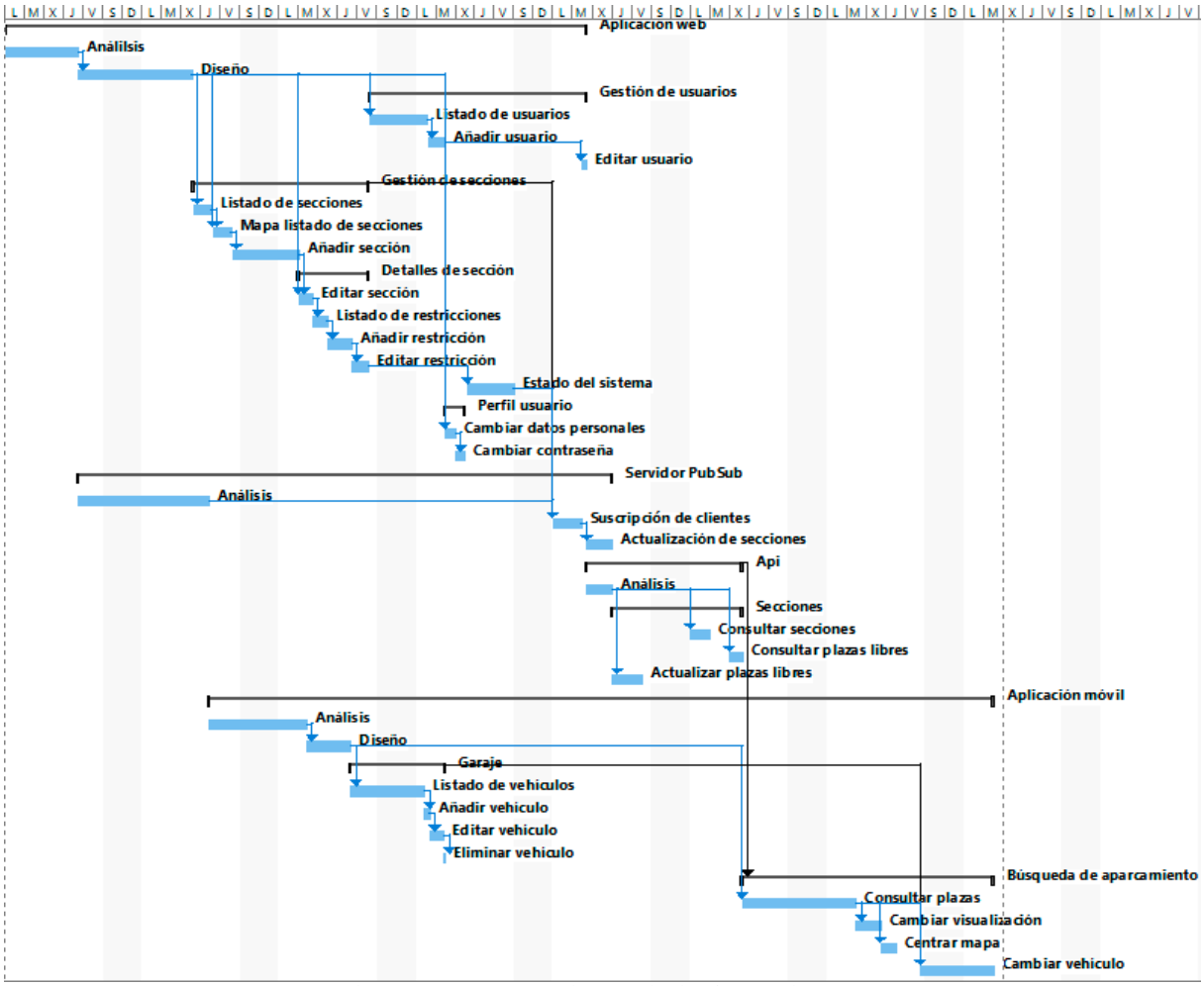


Ilustración 19 Diagrama de Gantt de planificación de tareas

9.2.RECURSOS HUMANOS

Rol	Horas	Salario/h	Total
Director de proyecto	60	60 €/h	3.600 €
Analista	99	45 €/h	4.455 €
Programador Senior web	148	35 €/h	5.180 €
Programado Android	78	35 €/h	2.730 €
Diseñador	50	25 €/h	1.250 €
Administrador de sistemas	50	35 €/h	1.750 €
		Total	**18.965 €**

9.3.COSTES ESTRUCTURALES

Descripción	**Coste**
Oficinas (4 meses)	8,000€
Equipos (5 PCs)	5.000€
Servidor Web	100€/mes

10. CONCLUSIONES

En este apartado explicaré las conclusiones obtenidas después del desarrollo del proyecto. Estas conclusiones se dividen en las obtenidas en la realización del proyecto, en las relativas a las herramientas utilizadas y en las personales.

10.1. PROYECTO

Después de finalizar el proyecto y de utilizar virtualmente la aplicación existen varias formas en que pueda ser aplicada dependiendo de la situación en cada región.

En regiones con **muy pocas plazas libres y con horario de regulación** serviría para tomar decisiones de que dirección escoger para seguir buscando plazas libres. Aunque el tiempo de que una plaza permanece libre sea muy efímero, existe una alta rotación de vehículos provocado por la regulación de horarios. Si como máximo un vehículo puede estar 2 horas en una zona azul, y en una calle media de Barcelona pueden haber unos 12 vehículos, significa que cada 2 horas se han de renovar esos 12 vehículos con lo que cada 10 minutos se queda una plaza libre. Este sería el caso por ejemplo del centro de Barcelona, donde hay muy pocas plazas libres y la mayoría reguladas, con lo que obliga a una rotación.

En regiones con **muy pocas plazas y sin horario de regulación** serviría para tomar decisiones de que dirección escoger o dejar el vehículo en un parking privado, o incluso detener el vehículo en una zona segura a la espera que haya movimiento. Este sería el caso por ejemplo del Campus Nord de la UPC, donde existen muy pocas plazas libres y al no haber ninguna regulación, los vehículos pueden permanecer aparcados durante todo el día sin moverse.

En regiones con **relativa facilidad de aparcamiento** serviría para poder tomar la decisión de aparcar en una zona más cercana a la que nos dirigimos, el usuario conoce de antemano si, por ejemplo, en la puerta del lugar al que se dirige existe una plaza libre, de esta manera se acercaría más a su destino, en cambio si sabe que no existe ninguna plaza libre y encuentra una un par de calles antes, puede tomar la decisión de aparcar directamente en ese punto, eliminando el vehículo del tráfico.

En regiones **con muchas plazas libres** pasaría como en el caso anterior, aunque realmente en éstas no sería necesario el uso de la aplicación ya que no existe un problema que resolver.

10.2. HERRAMIENTAS

Después de haber trabajado con las herramientas de *Symfony*, *MongoDB* y *Android* por primera vez puedo sacar las siguientes conclusiones:

10.2.1. Symfony

Symfony permite ahorrar mucho tiempo en el desarrollo gracias a sus componentes y a su completa integración con componentes de terceros mediante *Composer*. Los problemas que me he encontrado con Symfony han sido sobre todo relacionados con mi falta de experiencia, lo cual ha hecho que cosas sencillas consumieran mucho tiempo en encontrar cómo solucionarlas.

También hay que pensar que no solo se trabaja con Symfony, sino con todas las dependencias que requiera nuestro proyecto, por lo que puede que nos encontremos con algún problema en estas dependencias, ya sea porque tenga algún error o lo que es más común, que no tenga una documentación completa de sus funcionalidades.

Es importante antes de incorporar una dependencia a nuestro proyecto consultar si el proyecto sigue activo, ya que si es así, nos dará menos problemas. Además si es un proyecto *Open Source* será más fácil obtener ayuda de la comunidad e incluso contribuir al proyecto.

Durante la realización de este trabajo se contribuyó en el proyecto *CiRestClientBundle* notificando una dependencia no establecida en su configuración a través de la página de *GitHub*: https://github.com/CircleOfNice/CiRestClientBundle/pull/38.

También se comunicó un error en el proyecto *mongodb-odm* en el uso de *CustomCollections*, el cual era provocado por que no estaba integrado aún el uso de *CustomCollections* en el proyecto principal, con lo que amablemente, los desarrolladores actualizaron las dependencias para que funcionase correctamente. https://github.com/doctrine/mongodb-odm/issues/1415

Al finalizar el proyecto puedo decir que he aprendido bastante en el desarrollo con *Symfony*, aunque aún me queda mucho por aprender, creo que he adquirido un nivel adecuado

para poder seguir utilizándolo como *framework* de programación para mis próximos proyectos personales y profesionales.

10.2.2. MongoDB

Estoy acostumbrado a trabajar con base de datos relacionales SQL, como MySQL y PostrgreSQL, en las cuales tengo bastante experiencia, con lo que trabajar en un sistema de bases de datos NoSQL como MongoDB no es intuitivo y me ha resultado bastante complicado.

Los documentos anidados tienen su utilidad y un ahorro considerable de velocidad de acceso, pero consultar sobre los mismos es muy complicado, al menos con mi falta de experiencia en este sistema.

También echo en falta un lenguaje de consulta como *SQL*, sin el cual se hace complicado empezar a trabajar con este sistema para los que tenemos experiencia en bases de datos *SQL*.

10.2.3. Android

Para la realización de este trabajo he tenido que aprender a programar en *Android*, del que solo poseía nociones básicas que adquirí hace algunos años y que no me dotaban del nivel necesario para afrontar este proyecto.

Por este motivo, durante este año, he asisto a un un curso en la universidad para actualizarme y aprender la base para realizar la aplicación. Tengo bastante experiencia programando en *Java* con lo que adaptarme al entorno de trabajo no ha sido complicado.

Los problemas que encuentro a la hora de trabajar en *Android* han sido derivados más que nada del lenguaje *Java*. El problema de *Java* es que muchas veces para desarrollar algo muy sencillo debes realizar mucho código, en contrapartida con otros lenguajes interpretados que hacen el trabajo más fácil, claro que esto permite una mayor eficiencia en el código en el momento de la ejecución.

También el uso de *Android Studio* ha ayudado, ya que tener un entorno de desarrollo fácil de instalar y de configurar permite empezar a trabajar desde el primer momento y no perder tiempo instalando diferentes aplicaciones. Antes, conseguir instalar y configurar el entorno para desarrollar en Android, era bastante más complicado.

10.3. PERSONALES

A nivel personal, considero que el proyecto me ha ayudado a poner en práctica los conocimientos adquiridos en la facultad, y a resucitar muchas cosas, que en la vida profesional, nos saltamos para agilizar el trabajo y reducir costes.

Además, este proyecto me ha dado la oportunidad de estudiar nuevas tecnologías que me permitirán aplicarlas a los nuevos proyectos que realice en mi carrera profesional.

10.4. COMPETENCIAS TRANSVERSALES

Durante la ejecución de este proyecto se han trabajado diferentes competencias transversales de la carrera de Ingeniería en Informática.

- **Espíritu emprendedor e innovador:** Este proyecto nace con la idea de convertirse en una realidad, en el desarrollo de un producto viable para a partir de él poder desarrollar un proyecto empresarial sostenible. Aunque existen ya ideas y desarrollos en la detección de plazas de aparcamiento, este proyecto pretende separar la generación de información de la consulta de la misma con un sistema único, permitiendo la convivencia de varios sistemas de detección independientes.
- **Sostenibilidad y compromiso social:** Este proyecto busca contribuir en la reducción de las emisiones contaminantes que afectan a las ciudades con un gran número de población, así como plantear la viabilidad económica tanto en el coste del proyecto como en los beneficios que se obtendrían tras su implantación. Además la idea es que todos los ciudadanos tengan acceso a esta herramienta e información sin ningún tipo de coste.
- **Comunicación eficaz oral y escrita:** En el desarrollo de esta memoria he tenido que analizar y sintetizar toda la información del desarrollo del proyecto para permitir una lectura fácil y comprensible, adaptando la redacción según el nivel técnico del contenido en cada punto.
- **Uso solvente de los recursos de información:** Para desarrollar el Análisis de impacto económico y ambiental (1.1) y el estado del arte (4) he realizado una

tarea de investigación, obteniendo información diversa de diferentes fuentes. He analizado cada fuente y sintetizado conjuntamente la información que aportan para incluirlas en el proyecto.

- **Aprendizaje autónomo:** Durante la fase de implementación de este proyecto he trabajado tecnologías nuevas para mí como *MongoDB*, *Symfony* y *Android*. Todo el aprendizaje ha sido a base de buscar información y testear ejemplos. He buscado la mejor solución en cada punto del desarrollo después de analizar y testear diversas opciones.
- **Hábitos de pensamiento:** Durante el desarrollo del proyecto, he tenido que investigar y testar diferentes tecnologías para poder utilizarlas dentro del proyecto. En algunos casos incluso se ha descartado algún componente que más tarde se ha vuelto reevaluar y ha acabado dentro del proyecto.

11. PROPUESTAS DE MEJORAS

En este apartado se comentaran algunas de las mejoras que se pueden aplicar al proyecto tanto para dotarlo de nuevas funcionalidades como para aumentar su seguridad y disponibilidad en un entorno de producción.

11.1. Escalabilidad:

Se debe permitir que el sistema sea escalable. Actualmente un único servidor concentra toda la carga de peticiones. Para realizar esta escalabilidad se debe:

1- Base de datos: Se debería crear un clúster de servidores para *MongoDB* y escalar según demanda. Además se puede utilizar la geo posición de las secciones y cubos para utilizar como *Shard key* e indexar las secciones en clústeres por ubicación geográfica.
2- Api y web: Tanto la aplicación web como la API se pueden escalar mediante un balanceador de carga que remita las peticiones a diferentes servidores web según s u carga.
3- WAMP: La aplicación WAMP se puede escalar mediante un balanceador de carga que redirija las peticiones de conexión a diferentes servidores según el cubo al que se quiera conectar. Siendo cada servidor responsable de un número determinado de cubos. Esta parte requerirá mayores cambios en el código para permitir a la aplicación establecer diversas conexiones a diversos servidores.

11.2. SEGURIDAD

1- SSL: Utilizar conexiones cifradas tanto para la aplicación web, la API y el servidor PubSub. El uso de SSL en la aplicación web, en la API y en el servidor PubSub serían fáciles de implementar sin cambiar apenas código. El uso de SSL en la aplicación Android requeriría el cambio de la librería utilizada actualmente para conectar al servidor PubSub ya que la actual no soporta SSL en la conexión a WebSockets.

2- Restringir el acceso a la API: Actualmente el acceso a la API no está protegido. En un entorno real en el que los sensores enviasen información a la API, ésta debería ofrecer algún tipo de autentificación para garantizar que el origen de la petición es de un sensor de confianza.
3- Restringir el acceso al servidor PubSub a usuarios autentificados y controlar las consultas realizadas para evitar sobrecarga de peticiones.

11.3. APLICACIÓN MÓVIL

1- Añadir autentificación de usuarios y guardar la configuración de cada usuario en un servicio centralizado.
2- Interacción verbal: Permitir que la aplicación comunique al usuario de forma verbal la aparición de nuevos espacios para aparcar así como si el espacio al que se dirige ha sido ocupado y recibir instrucciones mediante comandos por voz. Como por ejemplo:
    - Aplicación: Hay una plaza en la calle Barcelona número 2, ¿deseas ir ahí?
    - Usuario: Sí
    - Aplicación: Toma la primera salida en la próxima rotonda…
3- Detección automática de vehículo: Vincular y seleccionar automáticamente el vehículo al detectar el bluetooth.
4- Base de datos de vehículos: Añadir una base de datos de vehículos con sus medidas para que el usuario solo tenga que seleccionar el vehículo y automáticamente se detecte el largo y ancho del mismo.
5- Pago automático: Añadir la opción de pagar la estancia del vehículo en zonas de regulación conectando con los sistemas pertinentes de los ayuntamientos, permitiendo así en una única aplicación realizar los pagos y tener un historial de pagos realizados.

11.4. APLICACIONES DE INTELIGENCIA ARTIFICIAL

En la ejecución de este proyecto, se pueden aplicar diversas técnicas de inteligencia artificial para añadir nuevas capacidades al sistema.

11.4.1. Planificación de gestión de plazas

Se puede desarrollar un sistema que gestione y asigne plazas a los usuarios que están buscando aparcamiento en una misma zona. Actualmente el sistema muestra las plazas libres y el usuario decide a que plaza quiere dirigirse. El sistema podría, sabiendo a dónde se dirige cada usuario, asignar y reservar una plaza para cada usuario, optimizando la distancia de la plaza al destino del usuario y el tiempo que llevan los usuarios buscando plazas.

Se podrían utilizar aplicaciones ya desarrolladas como *OptaPlanner*[25] de *Redhat* o desarrollar un producto a medida.

11.4.2. Análisis de datos

El sistema genera una gran información a cada momento, si se tiene toda una ciudad con sensores y los usuarios hacen uso de la aplicación, la cantidad de información generada de oferta y demanda puede ser abismal.

Con esta información se podrían aplicar técnicas de *machine learning*[26] y *big data*[27] para extraer conclusiones valiosas para la gestión de movilidad de una ciudad como pudiera ser la añadir, retirar, o cambiar zonas reguladas, definir nuevas rutas o alterar las ya establecidas de transporte público, gestión inteligente del tráfico mediante el uso de semáforos variables e incluso replantear la circulación del tráfico.

11.4.3. Vehículos autónomos

Los vehículos autónomos son, cada vez más, una realidad, y la ciencia está avanzando rápidamente hacia su implantación en entornos reales. Este sistema permitiría a los vehículos autónomos ver más allá de su visión local permitiendo a un vehículo por ejemplo encontrar aparcamiento por sí mismo.

También permitiría tener un vehículo autónomo a modo de taxi personal, de tal manera que el vehículo deja al usuario en la puerta de casa y por sí mismo luego se dirige a encontrar aparcamiento y, al día siguiente, vuelve a buscar al usuario a la puerta de su casa, como el sistema *Summon*[28] de *Tesla* pero aplicado a plazas de aparcamiento públicas.

---

[25] Solucionador de satisfacción de restricciones: http://www.optaplanner.org/
[26] https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico
[27] https://es.wikipedia.org/wiki/Big_data
[28] https://www.tesla.com/blog/summon-your-tesla-your-phone

También los propios vehículos podrían utilizarse como sensores móviles en la ciudad, siendo estos mismos los que detecten plazas de aparcamiento y las notifiquen al sistema central, ahorrando mucho dinero a una ciudad al no necesitar la implantación de sensores, utilizando visión por computador[29] y técnicas de *Deep Learning*[30] para el procesamiento de imágenes que pueda captar y determinar si lo que está captando es un aparcamiento válido. Pero esto solo sería posible si la mayoría de vehículos son capaces de realizar esta tarea.

Incluso podría no haber ningún sistema central, y que los vehículos movieran la información entre ellos como una red P2P[31], haciendo que todos los vehículos de una ciudad pertenezcan a un gran sistema de información distribuido y dinámico como un sistema colaborativo de la Internet de las Cosas (IoT)[32].

[29] https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial
[30] https://es.wikipedia.org/wiki/Aprendizaje_profundo
[31] https://es.wikipedia.org/wiki/Peer-to-peer
[32] https://es.wikipedia.org/wiki/Internet_de_las_cosas

12. ANEXOS

A. DEFINICIONES

Sección (Section)

Se considera una sección a un tramo continuo en el que en parte del tramo se pueden estacionar vehículos en una única formación.

Cada sección se define como:

- Nombre (nombre de calle, intersección, zona)
- Camino: Sucesión de puntos geodésicos que conforman el tramo.
- Formación (línea, batería, diagonal, diagonal-inversa)

Se considerará punto de inicio el primer punto del tramo en el sentido de la vía en la que se encuentra el tramo.

Restricciones (Restrictions)

Cada tramo dispone de una serie de restricciones a lo largo de sí mismo. Estas restricciones están definidas por la posición que ocupan del tramo, el tipo de restricción y las fechas de aplicación de dicha restricción.

Por definición una restricción fuera de su horario de regulación se considera una zona hábil para el uso, así como para aquellos vehículos habilitados para utilizarla.

- Posición inicial (en centímetros)
- Distancia de la restricción
- Fechas aplicables.
- Tipo de restricción (vado, carga y descarga, minusválidos, zona azul/verde, etc.)

Cubo (Bucket)

Se considera un cubo como una región rectangular que contiene diversas secciones. Cada sección pertenece a un único cubo.

Una sección puede empezar y acabar en dos cubos diferentes siempre y cuando estos dos cubos sean adyacentes.

B. SOFTWARE

En este apartado se muestra el software utilizado tanto para desarrollar la aplicación como el software dependiente de la aplicación desarrollada. Para las dependencias de la aplicación s e incluyen las versiones utilizadas y las licencias de cada una.

Herramientas del proyecto

Las herramientas del proyecto son el software utilizado para desarrollar el proyecto, tanto las aplicaciones como la memoria. Todas las herramientas podrían ser cambiadas por otras herramientas que cumpliesen el mismo objetivo.

Windows 10: Windows 10 ha sido el sistema operativo utilizado para desarrollar el proyecto. Cualquier otro sistema operativo hubiera sido perfectamente válido.

VMWare: Software de virtualización de sistemas operativos. Utilizado para virtualizar el servidor *Ubuntu*. Se podría haber utilizado *VirtualBox* o una máquina física.

Ubuntu: Sistema operativo sobre el que se ha ejecutado la aplicación. Cualquier otra distribución de Linux podría haber sido utilizada.

Microsoft Office: Suite ofimática para el desarrollo de la documentación. Se podría haber utilizado cualquier otro software como *Libre Office* o *LaTeX*.

Sublime Text: Editor de archivos para desarrollar la programación. Cualquier editor de archivos podría haber sido utilizado.

Android Studio: IDE de programación para Android. Se podría haber utilizado cualquier otro IDE que fuese compatible, como Eclipse.

Visual Paradigm: Herramienta para diseñar UML. Se podría haber utilizado cualquier otra herramienta.

Git: Herramienta de control de versiones.

Gitlab: Gestor de repositorios *git* con interfaz web.

Microsoft Project: Herramienta para planificación de proyectos. Se podría utilizar *OpenProject*.

Dependencias de las aplicaciones

Las dependencias de las aplicaciones son librerías, herramientas o aplicaciones que son dependencias directas del software desarrollado, estas dependencias podrían o no cambiarse por otras. En caso de cambiarlas requeriría de un cambio en la programación.

Aplicación Web, API, simulador y servidor PubSub

Nombre	Apache2
Versión	2.4.7
Url	https://httpd.apache.org/
Licencia	APACHE LICENSE, VERSION 2.0
Descripción	Servidor web para procesar peticiones http.

Nombre	MariaDB
Versión	5.5.49
Url	https://mariadb.org/
Licencia	GNU General Public License, version 2
Descripción	Motor de base de datos relacional

Nombre	MongoDB
Versión	5.5.49
Url	https://www.mongodb.com/
Licencia	GNU General Public License, version 3.0
Descripción	Motor de base de datos NoSQL

Nombre	Symfony2
Versión	2.8.9
Url	https://symfony.com/
Licencia	MIT license
Descripción	Motor de base de datos NoSQL

Nombre	ZeroMQ
Versión	4.1.5
Url	http://zeromq.org/
Licencia	GNU Lesser General Public License
Descripción	Motor de distribución de mensajes

Nombre	Composer
Versión	1.1.2
Url	https://getcomposer.org/
Licencia	MIT license
Descripción	Gestor de dependencias para PHP

La aplicación web, la API y el simulador están desarrollados sobre Symfony2. A continuación se muestra la tabla de todos los paquetes instalados mediante *composer* con sus versiones y licencias. Los paquetes en negrita son dependencias directas, el resto son dependencias de las dependencias.

Paquete	**Version**	**Licencia**	**Descripción**
bmatzner/fontawesome-bundle	4.6.1	MIT, CC BY 3.0	FontAwesome Bundle for Symfony2

ci/restclientbundle	2.0.0	GPL	Provides REST request methods. Mapper bundle for php internal curl library
doctrine/annotations	v1.2.7	MIT	Docblock Annotations Parser
doctrine/cache	v1.6.0	MIT	Caching library offering an object-oriented API for many cache backends
doctrine/collections	v1.3.0	MIT	Collections Abstraction library
doctrine/common	v2.6.1	MIT	Common Library for Doctrine projects
doctrine/dbal	v2.5.4	MIT	Database Abstraction Layer
doctrine/doctrine-bundle	1.6.3	MIT	Symfony DoctrineBundle
doctrine/doctrine-cache-bundle	1.3.0	MIT	Symfony Bundle for Doctrine Cache
doctrine/inflector	v1.1.0	MIT	Common String Manipulations with regard to casing and singular/plural rules.
doctrine/instantiator	1.0.5	MIT	A small, lightweight utility to instantiate objects in PHP without invoking their constructors
doctrine/lexer	v1.0.1	MIT	Base library for a lexer that can be used in Top-Down, Recursive Descent Parsers.
doctrine/mongodb	1.3.0	MIT	Doctrine MongoDB Abstraction Layer
doctrine/mongodb-odm	1.1.1	MIT	Doctrine MongoDB Object Document Mapper
doctrine/mongodb-odm-bundle	3.2.0	MIT	Symfony2 Doctrine MongoDB Bundle
doctrine/orm	v2.5.4	MIT	Object-Relational-Mapper for PHP
evenement/evenement	v2.0.0	MIT	Événement is a very simple event dispatching library for PHP
friendsofsymfony/user-bundle	dev-master	MIT	edbf840 Symfony FOSUserBundle
gos/pnctl-event-loop-emitter	v0.1.5	MIT	Pnctl event emitter for event loop
gos/pubsub-router-bundle	v0.2.1	MIT	Symfony PubSub Router Bundle

gos/ratchet	v0.3.5	MIT	PHP WebSocket library
gos/ratchet-stack	v0.1.0	MIT	Ratchet Stack
gos/web-socket-bundle	v1.8.3	MIT	Symfony Web Socket Bundle
gos/websocket-client	v0.1.2	MIT	WAMP client in PHP
guzzle/common	v3.9.2	MIT	Common libraries used by Guzzle
guzzle/http	v3.9.2	MIT	HTTP libraries used by Guzzle
guzzle/parser	v3.9.2	MIT	Interchangeable parsers used by Guzzle
guzzle/stream	v3.9.2	MIT	Guzzle stream wrapper component
incenteev/composer-parameter-handler	v2.1.2	MIT	Composer script handling your ignored parameter file
ircmaxell/password-compat	v1.0.4	MIT	A compatibility library for the proposed simplified password hashing algorithm: https://wiki.php.net/rfc/password_...
jdorn/sql-formatter	v1.2.17	MIT	a PHP SQL highlighting library
kriswallsmith/assetic	v1.3.2	MIT	Asset Management for PHP
monolog/monolog	1.21.0	MIT	Sends your logs to files, sockets, inboxes, databases and various web services
paragonie/random_compat	v2.0.2	MIT	PHP 5.x polyfill for random_bytes() and random_int() from PHP 7
psr/log	1.0.0	MIT	Common interface for logging libraries
react/event-loop	v0.4.2	MIT	Event loop abstraction layer that libraries can use for evented I/O.
react/socket	v0.4.3	MIT	Library for building an evented socket server.
react/stream	v0.4.3	MIT	Basic readable and writable stream interfaces that support piping.
react/zmq	v0.3.0	MIT	ZeroMQ bindings for React.
sensio/distribution-bundle	v5.0.7	MIT	Base bundle for Symfony Distributions
sensio/framework-extra-bundle	v3.0.16	MIT	This bundle provides a way to configure your controllers with

			annotations
sensio/generator-bundle	v3.0.7	MIT	This bundle generates code for you
sensiolabs/security-checker	v3.0.2	MIT	A security checker for your composer.lock
swiftmailer/swiftmailer	v5.4.3	MIT	Swiftmailer, free feature-rich PHP mailer
symfony/assetic-bundle	v2.8.0	MIT	Integrates Assetic into Symfony2
symfony/console	v3.1.3	MIT	Symfony Console Component
symfony/monolog-bundle	2.11.1	MIT	Symfony MonologBundle
symfony/phpunit-bridge	v2.8.9	MIT	Symfony PHPUnit Bridge
symfony/polyfill-apcu	v1.2.0	MIT	Symfony polyfill backporting apcu_* functions to lower PHP versions
symfony/polyfill-intl-icu	v1.2.0	MIT	Symfony polyfill for intl's ICU-related data and classes
symfony/polyfill-mbstring	v1.2.0	MIT	Symfony polyfill for the Mbstring extension
symfony/polyfill-php54	v1.2.0	MIT	Symfony polyfill backporting some PHP 5.4+ features to lower PHP versions
symfony/polyfill-php55	v1.2.0	MIT	Symfony polyfill backporting some PHP 5.5+ features to lower PHP versions
symfony/polyfill-php56	v1.2.0	MIT	Symfony polyfill backporting some PHP 5.6+ features to lower PHP versions
symfony/polyfill-php70	v1.2.0	MIT	Symfony polyfill backporting some PHP 7.0+ features to lower PHP versions
symfony/polyfill-util	v1.2.0	MIT	Symfony utilities for portability of PHP codes
symfony/process	v3.1.3	MIT	Symfony Process Component
symfony/security-acl	v3.0.0	MIT	Symfony Security Component - ACL (Access Control List)
symfony/swiftmailer-bundle	v2.3.11	MIT	Symfony SwiftmailerBundle
symfony/symfony	v2.8.9	MIT	The Symfony PHP framework

symfony/var-dumper	v3.1.3	MIT	Symfony mechanism for exploring and dumping PHP variables
twig/twig	v1.24.1	BSD-3-Clause	Twig, the flexible, fast, and secure template language for PHP

Aplicación Móvil

Nombre	Android SDK Tools
Versión	25.1.7
Url	https://developer.android.com/studio/releases/sdk-tools.html
Licencia	https://developer.android.com/studio/terms.html
Descripción	Set de desarrollo y depuración para Android SDK.

Nombre	Android Google Play Services
Versión	25.1.7
Url	https://developers.google.com/android/guides/overview
Licencia	-
Descripción	Api para uso de servicios de Google como Google Maps.

Nombre	Android Maps Utils
Versión	0.4
Url	https://github.com/googlemaps/android-maps-utils
Licencia	Apache License 2.0
Descripción	Librería para el uso de Google Maps.

Nombre	Android Maps Utils

Versión	0.4
Url	https://github.com/googlemaps/android-maps-utils
Licencia	Apache License 2.0
Descripción	Librería para el uso de Google Maps.

Nombre	Android Design
Versión	24.1.1
Url	http://android-developers.blogspot.com.es/2015/05/android-design-support-library.html
Licencia	Apache License 2.0
Descripción	Librería de soporte para el uso de Material Design

Nombre	Android Support
Versión	24.1.1
Url	https://developer.android.com/topic/libraries/support-library/index.html
Licencia	Apache License 2.0
Descripción	Librerías de soporte de Google

Nombre	Android CardView
Versión	24.1.1
Url	https://developer.android.com/training/material/lists-cards.html?hl=es
Licencia	Apache License 2.0

Descripción	Librería de soporte para el uso de CardView

Nombre	Android Recycler View
Versión	24.1.1
Url	https://developer.android.com/training/material/lists-cards.html?hl=es
Licencia	Apache License 2.0
Descripción	Librería de soporte para el uso de Recycler View

Nombre	Android Autobahn
Versión	0.5.2
Url	http://autobahn.ws/android/
Licencia	Apache License 2.0
Descripción	Librería de soporte para el uso de websockets y WAMP (Web Application Messaging Protocol)

Nombre	GSON
Versión	2.7
Url	https://github.com/google/gson
Licencia	Apache License 2.0
Descripción	Librería para convertir JSON en objetos Java y viceversa.

Nombre	Retrofit

Versión	2.1.0
Url	http://square.github.io/retrofit/
Licencia	Apache License 2.0
Descripción	Cliente HTTP de tipo seguro para Android y Java.

Nombre	Apache Commons Lang
Versión	3.3.4
Url	https://commons.apache.org/proper/commons-lang/
Licencia	Apache License 2.0
Descripción	Librería de utilidades para manipulación de cadenas de texto.

C. PUBSUB Y WAMP

El patrón de diseño publicación-subscripción (*PubSub*) es un patrón de comunicación dónde los publicadores (*publishers*) envían mensajes a los subscriptores (*subscribers*).

Los suscriptores se suscriben a temas (*topics*), y los publicadores publican mensajes en estos temas, de tal manera que los publicadores desconocen los usuarios que recibirán los mensajes. De esta manera se crea una capa de separación entre la publicación y la recepción de mensajes.

Wamp es un subprotocolo del estándar WebSocket que proporciona dos patrones de publicación mensajes basados en RPC y PubSub.

D. REALM JAVA

Realm es un sistema de gestión para bases de datos que permite trabajar como un ORM en dispositivos móviles.

Dispone de librerías para Java(Android), Objective-C, React-Native, Swift y Xamarin.

Configurar Realm en Android

Para poder utilizar Realm desde Android debemos añadir la dependencia y el plugin a nuestros ficheros gradle.

En el fichero *gradle* del proyecto debemos añadir dentro de la sección *buildscript* > *dependencies* la siguiente dependencia:

```
classpath "io.realm:realm-gradle-plugin:1.1.0"
```

Y acto seguido en nuestro fichero gradle de la aplicación añadir al principio del fichero el plugin de Realm.

```
apply plugin: 'realm-android'
```

Para configurar la base de datos simplemente ejecutamos el siguiente código en nuestro proyecto

```
RealmConfiguration realmConfig = new RealmConfiguration.Builder(c)
        .name(DATABSAE_NAME)
        .schemaVersion(DATABSAE_VERSION)
        .build();
Realm.setDefaultConfiguration(realmConfig);
```

Una vez configurada la base de datos podemos obtener el objeto Realm para poder ejecutar las operaciones.

```
Realm realm = Realm.getDefaultInstance();
```

Para convertir cualquier objeto y que pueda ser manejado por Realm debemos extender de la clase *RealmObject*. Mediante anotaciones sobre los métodos y variables se puede configurar el comportamiento de cada atributo del objeto. Por ejemplo si añadimos la notación @PrimaryKey a un atributo indicamos a Realm que ese atributo actuará como clave primaria, o si añadimos la anotación @Ignore permitimos que dicho atributo no se guarde en nuestra base de datos.

```
public class Vehicle extends RealmObject {
    @PrimaryKey
    private String id;
    private String name;

    /* Getters y setters */
}
```

Para poder consultar los vehículos de nuestra base de datos utilizaremos el objeto Realm indicando que entidad queremos recuperar mediante la función *where*:

```
Realm realm  = Realm.getDefaultInstance()
RealmResults<Vehicle> list = realm.where(Vehicle.class).findAll();
Vehicle v = realm.where(Vehicle.class).equalTo("id", id).findFirst();
```

Para añadir un objeto y añadirlo a nuestra base de datos Realm utilizaremos la función *copyToRealm.*

```
Vehicle v = new Vehicle(12, "Peugeot 207");
Realm realm  = Realm.getDefaultInstance()
realm.beginTransaction();
realm.copyToRealm(v);
realm.commitTransaction();
```

Para editar un objeto sólo tendremos que editar sus propiedades, siempre y cuando estemos entre un *begin/commitTransaction* estos cambios se guardarán en nuestra base de datos.

```
Realm realm  = Realm.getDefaultInstance()
Vehicle v = realm.where(Vehicle.class).equalTo("id", id).findFirst();
realm.beginTransaction();
v.setName("Peugeot 207 Plus");
realm.commitTransaction();
```

Para eliminar un objeto utilizaremos la función *deleteFromRealm.*

```
Realm realm  = Realm.getDefaultInstance()
Vehicle v = realm.where(Vehicle.class).equalTo("id", id).findFirst();
realm.beginTransaction();
v.deleteFromRealm();
realm.commitTransaction();
```

E. INSTALACIÓN Y CONFIGURACIÓN DE SERVIDOR

Instalación del software y necesario para un servidor *Ubuntu Server 16.04*.

```
$ apt-get update
$ apt-get -y dist-upgrade
```

Si el servidor dispone de menos de 4Gb de memoria RAM, instalar una partición swap.

```
$ fallocate -l 4G /swapfile
$ chmod 600 /swapfile
$ mkswap /swapfile
$ swapon /swapfile
```

Instalar Apache, *PHP 5.6* y *MariaDB* y crear la base de datos.

```
$ apt-get install -y apache2

$ apt-get install -y mariadb-server mariadb-client
$ mysql_secure_installation
$ mysql -u root -p -e "CREATE DATABASE delfos DEFAULT CHARACTER SET utf8;"
$ add-apt-repository ppa:ondrej/php
$ apt-get update
$ apt-get install python-software-properties php5.6 php5.6-mcrypt php5.6-mysql
php5.6-dev php5.6-xml php5.6-mbstring php5.6-curl libapache2-mod-php5.6
```

Instalar *MongoDB*

```
$ apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
$ echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.0 multiverse" |
sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list
$ sudo apt-get update
$ sudo apt-get install -y --allow-unauthenticated mongodb-org
```

Crear el fichero `/lib/systemd/system/mongod.service` con el siguiente contenido

```
[Unit]
Description=High-performance, schema-free document-oriented database
After=network.target
Documentation=https://docs.mongodb.org/manual

[Service]
User=mongodb
Group=mongodb
ExecStart=/usr/bin/mongod --quiet --config /etc/mongod.conf

[Install]
WantedBy=multi-user.target
```

Añadir *MongoDB* para ejecutar al arrancar la máquina y crear la base de datos.

```
$ systemctl start mongod
$ systemctl enable mongodb
$ mongo
MongoDB shell version: 3.0.12
connecting to: test
Welcome to the MongoDB shell.
> use delfos
switched to db delfos
> exit
```

Instalar extensión MongoDB para PHP.

```
$ apt-get install -y autoconf g++ make openssl libssl-dev libcurl4-openssl-dev pkg-
config libsasl2-dev libpcre3-dev
$ pecl install mongo
$ echo "extension=mongo.so" > /etc/php/5.6/mods-available/mongo.ini
$ phpenmod -v 5.6 mongo
$ service apache2 restart
```

Instalar ZMQ para la transmisión de mensajes.

```
$ wget https://github.com/zeromq/zeromq4-1/releases/download/v4.1.5/zeromq-
4.1.5.tar.gz
$ tar -xzvf zeromq-4.1.5.tar.gz
$ cd zeromq-4.1.5/
$ ./configure
$ make
$ make install
$ cd ..
$ rm -rf zeromq-4.1.5*
```

Instalar librería ZMQ para PHP

```
$ pecl install zmq-beta
$ echo "extension=zmq.so" > /etc/php/5.6/mods-available/zmq.ini
$ sudo phpenmod -v 5.6 zmq
$ service apache2 restart
```

Instalar composer y dependencias

```
$ php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
$ php -r "if (hash_file('SHA384', 'composer-setup.php') ===
'e115a8dc7871f15d853148a7fbac7da27d6c0030b848d9b3dc09e2a0388afed865e6a3d6b3c0fad45c
48e2b5fc1196ae') { echo 'Installer verified'; } else { echo 'Installer corrupt';
unlink('composer-setup.php'); } echo PHP_EOL;"
$ php composer-setup.php
$ mv composer.phar /usr/bin/composer.phar
$ rm composer-setup.php
$ apt-get install unzip
$ pecl install zip
$ echo "extension=zip.so" > /etc/php/5.6/mods-available/zip.ini
$ phpenmod -v 5.6 zip
```

Para configurar apache editar `/etc/apache/sites-available/000-default.conf` y cambiar el Document Root

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/delfos/web

        DirectoryIndex app.php

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Editar /etc/apache2/apache.conf y establecer AllowOverride a All en /var/www.

```
<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
</Directory>
```

Reiniciar apache para que los cambios tengan efecto

```
$ service apache2 restart
```

Copiar el código de la aplicación como usuario normal, para ello primero establecer los permisos a la carpeta `/var/www`.

```
$ chown [USER]:www-data /var/www
```

Descomprimir el contenido del fichero `server.tar.gz` en la carpeta `/var/www/delfos` y actualizar las dependencias. Una vez instaladas las dependencias el sistema nos pedirá los valores para la configuración del sistema.

```
$ cd /var/www/delfos
$ composer.phar update
Creating the "app/config/parameters.yml" file
Some parameters are missing. Please provide them.
database_host (127.0.0.1):
database_port (null):
database_name (delfos):
database_user (root):
database_password (null): ********
mailer_transport (smtp):
mailer_host (127.0.0.1):
mailer_user (null):
mailer_password (null):
secret (ThisTokenIsNotSoSecretChangeIt): ****************
mongodb_host ('mongodb://localhost:27017'):
mongodb_database (delfos):
pubsub_host ('public pubsub hostname/ip'): 192.168.18.133
pubsub_port (8080):
pubsub_local_host (127.0.0.1):
pubsub_local_port (5555):
api_url ('http://%pubsub_host%/delfos/web/app_dev.php/api'):
```

```
google_maps_api ('API key for Google Maps'):
$ php bin/console doctrine:schema:update --force
```

La Api Key para Google Maps la podemos obtener generando una clave en la consola API de Google (https://console.developers.google.com).

Para iniciar el servicio *PubSub* crear el archivo `/lib/systemd/system/delfos-pubsub.service` cambiando el usuario `[USER]` por el usuario web pertinente.

```
[Unit]
Description=Delfos PubSub server
After=mongod.service

[Service]
User=[USER]
Group=[USER]
ExecStart=/usr/bin/php /var/www/delfos/bin/console --env=prod
gos:websocket:server

[Install]
WantedBy=multi-user.target
```

Iniciar el servidor *PubSub* y añadirlo al inicio.

```
$ systemctl start delfos-pubsub
$ systemctl enable delfos-pubsub
```

En este punto ya podemos acceder a la página web introduciendo la dirección IP del servidor o su dominio en caso de haberlo configurado así en *apache*.

Para añadir un usuario y el privilegio de adminsitrador al panel de administración podemos realizarlo mediante la consola:

```
$ php bin/console fos:user:create
Please choose a username: david
Please choose an email: david.rojo@est.fib.upc.edu
Please choose a password:
Created user david
$ php bin/console fos:user:promote
Please choose a username: david
Please choose a role: ROLE_ADMIN
Role "ROLE_ADMIN" has been added to user "david".
```